European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D7.6

Large Scale Management Final Report

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
Jacobs University Bremen, JUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universitat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, CETIM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University of Surrey, UniS, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois - Campus scientifique
615, rue de Jardin Botanique - B.P. 101
F-54600 Villers Les Nancy Cedex
France
Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title:	Large Scale Management Final Report
Type:	Public
Editor(s):	Ramin Sadre
E-mail:	sadrer@cs.utwente.nl
Author(s):	WP7 Partners
Doc ID:	D7.6

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
0.1	2007-07-03	H. Tran, J. Schönwälder	Initial version of a LaTeX template
0.2	2009-12-03	R. Sadre	Initial version for integration
0.3	2009-12-03	R. Sadre	FMAD activity description
0.4	2009-12-24	Partners	Activity descriptions
0.5	2009-12-28	R. Sadre	Fixes

Contents

1 Executive Summary

Scalability is one of the major issues in network management. Many management solutions that are available for small networks cannot be easily applied to networks consisting of hundreds or thousands of communication entities. The objective of work package 7 is the collaborative research between the EMANICS partners in the field of large scale management with the goal to develop, evaluate and report new approaches and solutions in the area.

During 2009, the work in this work package was driven by five activities that started in January 2009 as a result of an open call for proposals. The activities were structured around two themes: security in large-scale environments and monitoring and configuration in large-scale environments. This deliverable reports the scientific results achieved in 2009 by those five activities. Note that this document supersedes the interim report D7.5.

2 Introduction

Scalability is one of the major issues in network management. Many management solutions that are available for small networks can not be easily applied to networks consisting of hundreds or thousands of communication entities. The research activities in this work package focus on the security, configuration and monitoring in large-scale environments. Work is structured around the following two tasks, that emerged from an open call for proposals:

- **T7.1 Security in large-scale environments:** Activities in this task address the design, implementation and evaluation of new techniques to manage the security in large-scale environments, comprising the detection of intrusions and attack attempts, the secure access to resources, etc.
- **T7.2 Monitoring and configuration in large-scale environments:** Large-scale networks pose several challenges for management, for example the large amount of traffic transported in such networks, the large and variable number of participating network entities and involved parties, etc. Activities in this task follow different approaches such as distribution and virtualization to allow the effective monitoring and configuration under such conditions.

An open call for activity proposals was initiated at the end of 2008 and finalized at the beginning of 2009. Five activities were chosen to be funded during the last phase of the EMANICS project.

In the following sections, the goals of those activities are described and the results achieved in the period of January 2009 – December 2009 are reported. For an overall view on the activities and the involved participants, we summarize them in the following:

- **BioScale II:** Extended Scalable Management of Biometric devices (UniZH, UniBwM)
- **FMAD:** Flow-based Monitoring and Anomaly Detection (UT, PSNC, JUB, UniZH, INRIA, UniBwM, UCL)
- **MaMANET:** Management of large MANETs (UniBwM, UZH, JUB)
- **SeMI:** Security Management Infrastructure (UniBw, Jacobs)
- **VirtMon:** Virtualization Monitoring (UPC, UCL, UniZH)

Note that this document supersedes the interim report D7.5.

3 Extended Scalable Management of Biometric devices (BioScale II)

3.1 Introduction

Biometric devices emerged to a mature technology and begin to fulfill its promises for physical access control and login to computers. But with today's device specific software provided by manufacturers, medium and large scale deployments of biometric devices is unfeasible. One of the main challenges of such large scale deployment projects is a scalable distribution system for biometric templates. The objective of the BioScale I and II activity is the development of a scalable management approach for biometric devices.

3.2 Activity Description

In this section the goals of the BioScale I and II approach will be described sketchy to give a general survey of the BioScale project as a whole:

3.2.1 BioScale I

As mentioned in the final report of BioScale I a scalable template distribution strategy for biometric devices in a large scale deployment has been defined and implemented. For this the template distributer of the application BioXes was reused and to make sure that the scalability tests were feasible the template distributer had to be changed into a prioritized template distributer. On the other hand a dummy device had to be implemented to make it possible to test the above mentioned distribution strategy. During the implementation of the initial BioScale I approach a set of additional requirements emerged, which have to be considered for a proper final design of a scalable distribution system in a large scale deployment of biometric devices. Therefore, to achieve a full-fledged and complete approach for a scalable management approach for biometric devices in a large scale environment, the BioScale II approach emerged.

3.2.2 BioScale II

As just mentioned additional requirements emerged in the BioScale I approach. Because of these newly detected requirements a small subset of existing tasks of the initial BioScale I approach had to be postponed and adopted to the BioScale II approach. For the BioScale II approach the following task have to be finished:

- Task 1: Set up test scenario and scalability tests.
- Task 2: Analysis of measurement results.
- Task 3: Comprehensible User Interface.
- Task 4: Distributed User Interface.

3.3 Strategy for BioScale II

In this section the tasks of the BioScale II approach are described in more detail and the strategy to fulfill the tasks is shown.

3.3.1 Task 1: Test scenario and scalability tests

Based on the BioScale I implementation's dummy device and its multiple instances the test scenario has to be defined and set up as well as tests have to be performed to show that the approach designed for the template distribution is scalable in terms of:

- time required for distribution
- bandwidth usage during distribution
- respond time of devices during template distribution
- integrity of distribution

3.3.2 Task 2: Analysis of measurement results

Those results obtained from the tests performed in Task 1 have to be analyzed. Based on these results of the analysis a policy for the template distribution in a large-scale environment has to be defined and fully implemented in turn.

3.3.3 Task 3: Comprehensible User Interface and Visualization

Especially in large-scale installations a comprehensive user interface is important in order to enable the user keeping the overlook. Designing such a user interface, which allows the user to manage a huge number of biometric devices and configure access rights for the distribution plan, respectively, is a challenging task. Therefore, the objective of task 2 is to design such a user interface and implement it prototypically for visualizing large-scale installations. In an iterative process the design of the interface will be improved by real-life tests.

3.3.4 Task 4: Distributed User Interface

Large-scale installations of biometric devices are mostly distributed, i.e., consisting of installations at different sites. The possibility of a distributed management is in this case of utmost importance. Centrally managing such installations would limit the scalability, not from a technical perspective, but from an organizational one. The key objective of this task is defined as designing relevant mechanisms for allowing several operators to administer biometric identities at the same time. The key technology to apply is to investigate Web 2.0 mechanisms and use those for biometric device and network management, where applicable. In case of functional gaps identified, different, proprietary solutions will be developed.

3.4 Final Report for BioScale I and BioScale II

3.4.1 Task 1: Set up the test scenario and perform the scalability test

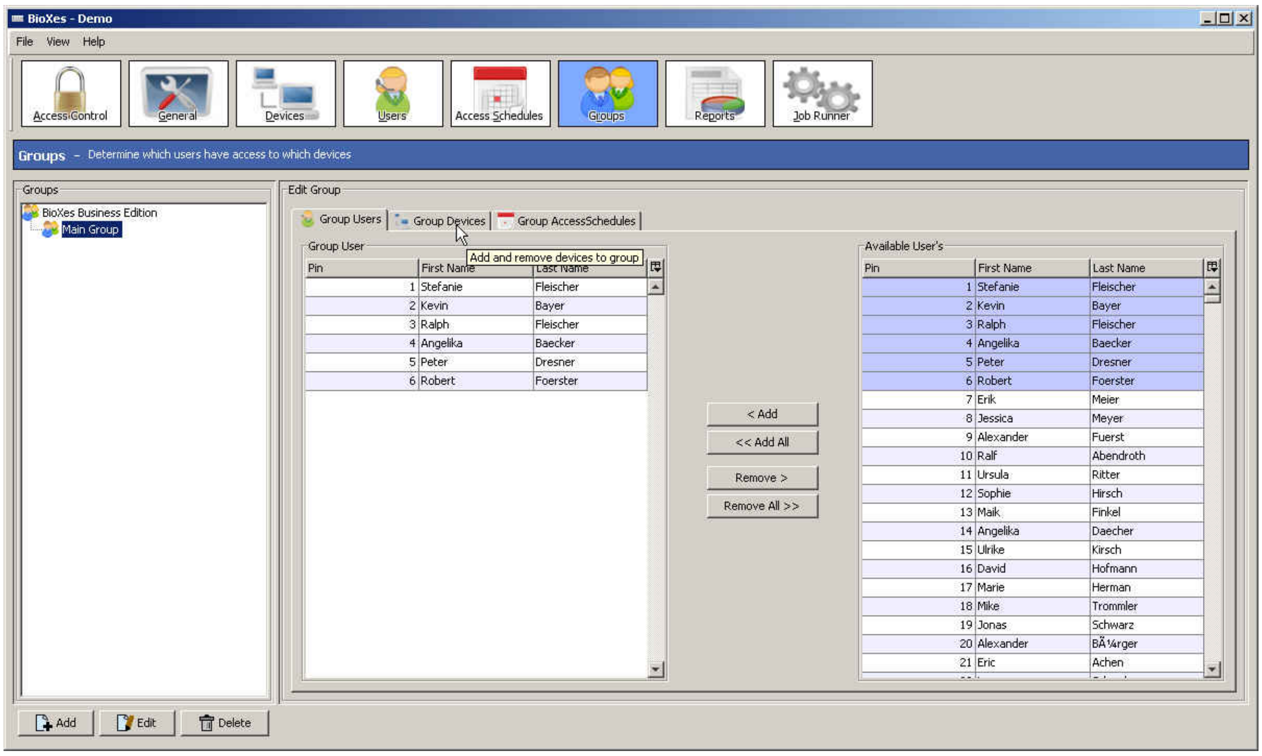
In order to have several thousand users to perform a real life test an LDAP integration has been implemented and tested. The virtual scenario with several thousand devices and users has not been set up, since more effort was spent on the Task 4. View scalability test have been carried out to survey if the template distributer performs its task. The result of the test was satisfying in a small test environment with several devices and users.

3.4.2 Task 2: Analyze measurement results

Since this task depends on the results of task 1 no action has taken place here.

3.4.3 Task 3: Comprehensible user interface and visualization

The final user interface including the second user feedback has been implemented and field tests have been carried out to assure that all functionality is provided. The resulting group screen has changed as follows:



Group User

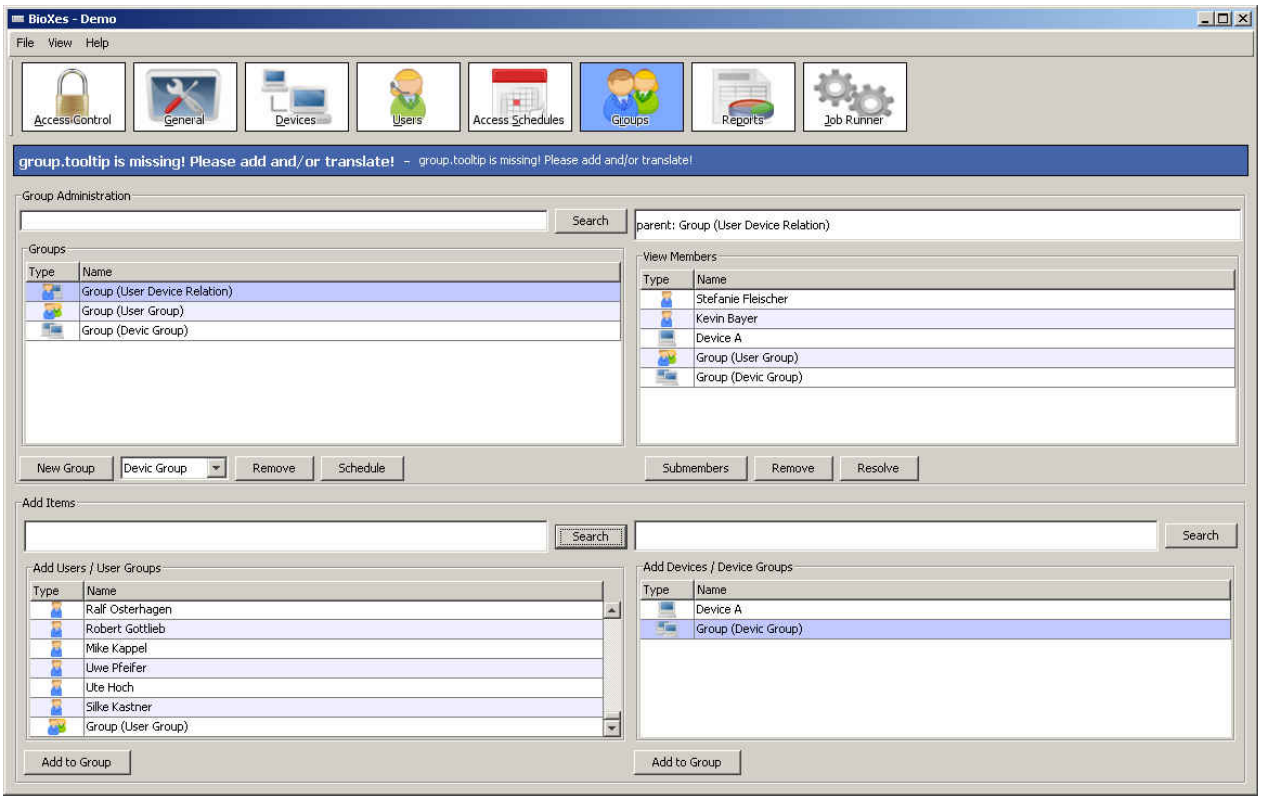
Pin	First Name	Last Name
1	Stefanie	Fleischer
2	Kevin	Bayer
3	Ralph	Fleischer
4	Angelika	Baecker
5	Peter	Dresner
6	Robert	Foerster

Available User's

Pin	First Name	Last Name
1	Stefanie	Fleischer
2	Kevin	Bayer
3	Ralph	Fleischer
4	Angelika	Baecker
5	Peter	Dresner
6	Robert	Foerster
7	Erik	Meier
8	Jessica	Meyer
9	Alexander	Fuerst
10	Ralf	Abendroth
11	Ursula	Ritter
12	Sophie	Hirsch
13	Maik	Finkel
14	Angelika	Daecher
15	Ulrike	Kirsch
16	David	Hofmann
17	Marie	Herman
18	Mike	Trommler
19	Jonas	Schwarz
20	Alexander	BÃ¼rger
21	Eric	Achen

Figure 1: In this picture the old version of the user interface of the distribution plan is shown.

The final version of the group screen will provide both views to the user, the old and new screen. This is because in second round of field tests some users performed better with the old screen and some with the new screen. Also some users even mentioned to have both screens available helped them to have a better overview of the template distribution plan.



group.tooltip is missing! Please add and/or translate! – group.tooltip is missing! Please add and/or translate!

parent: Group (User Device Relation)

Groups

Type	Name
	Group (User Device Relation)
	Group (User Group)
	Group (Devic Group)

View Members

Type	Name
	Stefanie Fleischer
	Kevin Bayer
	Device A
	Group (User Group)
	Group (Devic Group)

Add Users / User Groups

Type	Name
	Ralf Osterhagen
	Robert Gottlieb
	Mike Kappel
	Uwe Pfeifer
	Ute Hoch
	Silke Kastner
	Group (User Group)

Add Devices / Device Groups

Type	Name
	Device A
	Group (Devic Group)

Figure 2: In this picture the second version of the user interface of the distribution plan is shown.

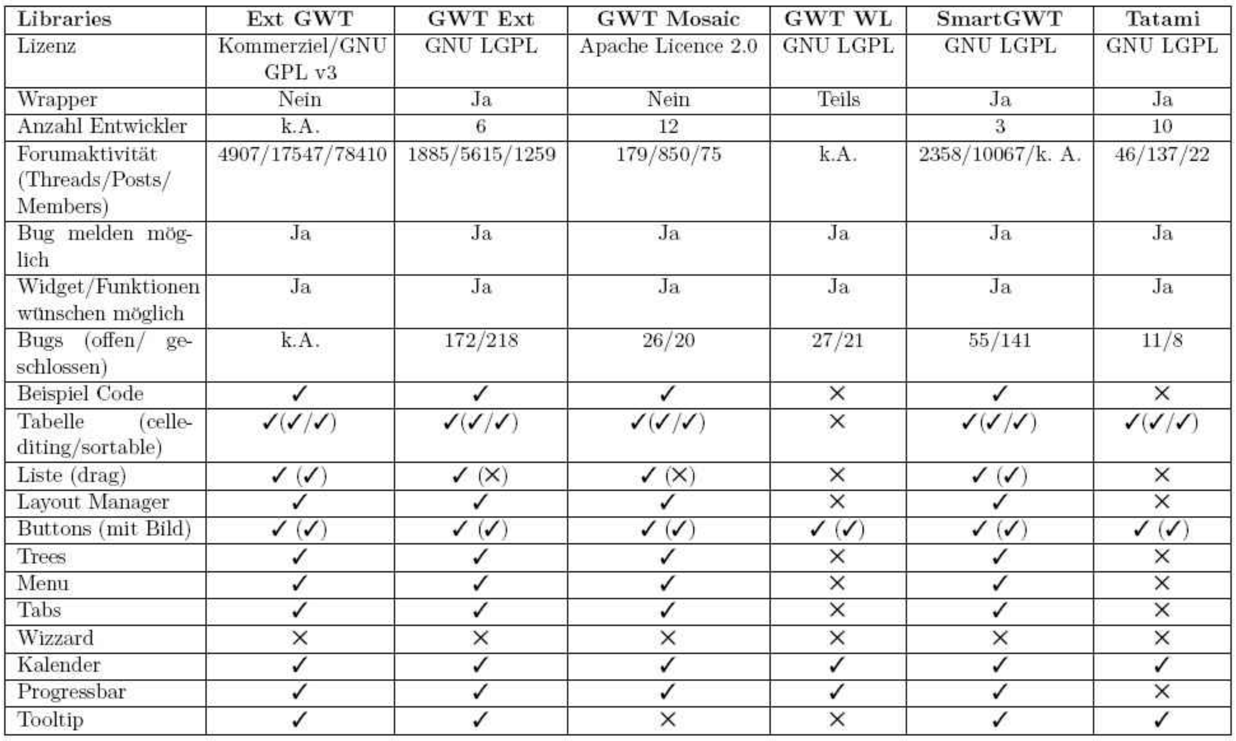
3.4.4 Task 4: Distributed User Interface

The distributed user interface has been implemented using the GWT-Mosaic framework. The GWT-Mosaic framework was chosen after a detailed evaluation of different available GWT widget libraries as shown in Figure 3.

With the new GWT version of BioXes the scalability of the application is given and allows several operators to administer biometric identities at the same time, independent of the locality.

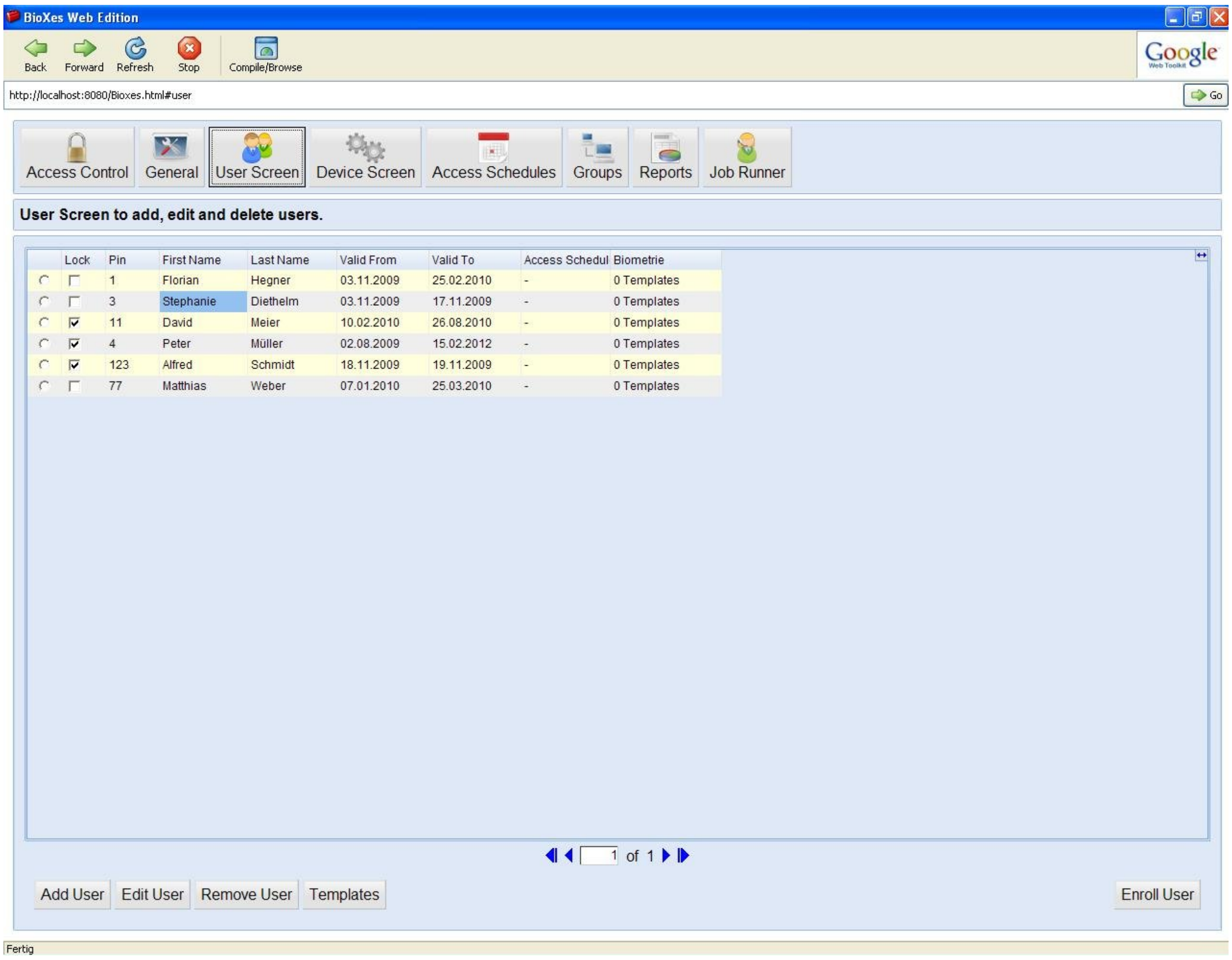
3.5 Further Work for BioScale

Set up virtual scenario with several thousand devices and users and analyze the measurement results.



Libraries	**Ext GWT**	**GWT Ext**	**GWT Mosaic**	**GWT WL**	**SmartGWT**	**Tatami**
Lizenz	Kommerziel/GNU GPL v3	GNU LGPL	Apache Licence 2.0	GNU LGPL	GNU LGPL	GNU LGPL
Wrapper	Nein	Ja	Nein	Teils	Ja	Ja
Anzahl Entwickler	k.A.	6	12		3	10
Forumaktivität (Threads/Posts/ Members)	4907/17547/78410	1885/5615/1259	179/850/75	k.A.	2358/10067/k. A.	46/137/22
Bug melden möglich	Ja	Ja	Ja	Ja	Ja	Ja
Widget/Funktionen wünschen möglich	Ja	Ja	Ja	Ja	Ja	Ja
Bugs (offen/ geschlossen)	k.A.	172/218	26/20	27/21	55/141	11/8
Beispiel Code	✓	✓	✓	×	✓	×
Tabelle (celleditiing/sortable)	✓(✓/✓)	✓(✓/✓)	✓(✓/✓)	×	✓(✓/✓)	✓(✓/✓)
Liste (drag)	✓ (✓)	✓ (×)	✓ (×)	×	✓ (✓)	×
Layout Manager	✓	✓	✓	×	✓	×
Buttons (mit Bild)	✓ (✓)	✓ (✓)	✓ (✓)	✓ (✓)	✓ (✓)	✓ (✓)
Trees	✓	✓	✓	×	✓	×
Menu	✓	✓	✓	×	✓	×
Tabs	✓	✓	✓	×	✓	×
Wizzard	×	×	×	×	×	×
Kalender	✓	✓	✓	✓	✓	✓
Progressbar	✓	✓	✓	✓	✓	×
Tooltip	✓	✓	×	×	✓	✓

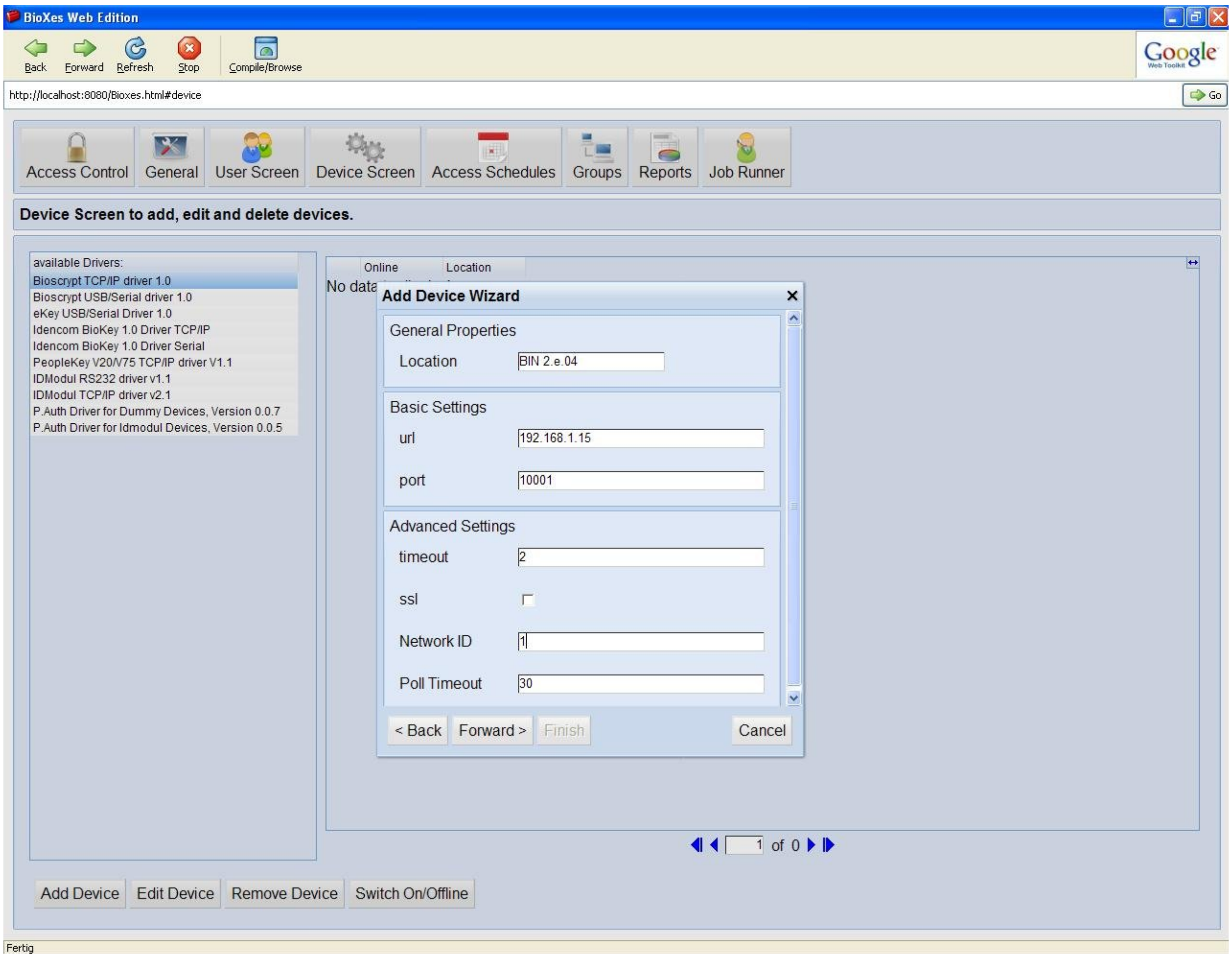
Figure 3: Overview of GWT widget libraries



User Screen to add, edit and delete users.

	Lock	Pin	First Name	Last Name	Valid From	Valid To	Access Schedul	Biometrie
○	☐	1	Florian	Hegner	03.11.2009	25.02.2010	-	0 Templates
○	☐	3	Stephanie	Diethelm	03.11.2009	17.11.2009	-	0 Templates
○	☑	11	David	Meier	10.02.2010	26.08.2010	-	0 Templates
○	☑	4	Peter	Müller	02.08.2009	15.02.2012	-	0 Templates
○	☑	123	Alfred	Schmidt	18.11.2009	19.11.2009	-	0 Templates
○	☐	77	Matthias	Weber	07.01.2010	25.03.2010	-	0 Templates

Figure 4: User Screen: the distributed user screen allows adding, editing and deleting users. Furthermore users can be enrolled using the “Enrol User” button.



http://localhost:8080/Bioxes.html#device

Device Screen to add, edit and delete devices.

available Drivers:
- Bioscrypt TCP/IP driver 1.0
- Bioscrypt USB/Serial driver 1.0
- eKey USB/Serial Driver 1.0
- Idencom BioKey 1.0 Driver TCP/IP
- Idencom BioKey 1.0 Driver Serial
- PeopleKey V20/V75 TCP/IP driver V1.1
- IDModul RS232 driver v1.1
- IDModul TCP/IP driver v2.1
- P.Auth Driver for Dummy Devices, Version 0.0.7
- P.Auth Driver for Idmodul Devices, Version 0.0.5

No data

Add Device Wizard

General Properties

Location	BIN 2.e.04

Basic Settings

url	192.168.1.15
port	10001

Advanced Settings

timeout	2
ssl	☐
Network ID	1
Poll Timeout	30

Figure 5: Device Screen: in the distributed device screen all by BioXes supported biometric devices is listed. The screen is used to add, edit and delete devices.

4 Flow-based Monitoring and Anomaly Detection (FMAD)

4.1 Introduction and activity description

The spread of Gbps networks and the constantly increasing amount of traffic transported by such networks has drawn the attention to scalability aspects of the management of such networks, leading to an interest in flow-based techniques for analysis and anomaly detection in large and/or fast networks. In this activity, we have developed and evaluated several techniques and approaches that allow to characterize and analyze flow data collected in such large-scale, resp. high-speed networks.

Section 4.2 provides a survey of current research in the area of flow-based intrusion detection. The survey starts with a motivation why flow-based intrusion detection is needed. The concept of flows is explained, and relevant standards are identified. We provide a classification of attacks and defense techniques and show how flow-based techniques can be used to detect scans, worms, Botnets and Denial-Of-Service (DoS) attacks. The survey has been accepted for publication and will appear in [1].

In Section 4.3, it is investigated if spammers can be detected at the network level, based on just flow data. This problem is challenging, since no information about the content of the email message is available. A spam detection algorithm is proposed, which is able to discriminate between benign and malicious hosts with high accuracy. Please note that the achieved results have been published in [2].

In Section 4.4, we describe the modeling of SSH brute force attacks by Hidden Markov models. Such models can be used to generate synthetic traffic traces for the design and evaluation of anomaly detection algorithms. The described work has been published in [3].

In Section 4.5, we analyze international network traffic of a research network, the PIONIER Polish Optical Internet, using two different approaches (sampled NetFlow and optical passive monitoring) and discuss the results obtained by them.

One objective when investigating traces is to detect traffic regularities such as repeating patterns, which can be associated with the usage of common network services. This approach can be further extended to detect traffic irregularities such as network anomalies or attacks, which also generate specific patterns. These patterns typically spread over several flows. In Section 4.6, we propose a flow record query language, which allows to describe patterns in a declarative and easy to understand way.

Section 4.7 discusses how new approaches like Network Behavior Analyses (NBA) Systems show promise for being able to cope with the new threats. These systems evaluate statistical flow data generated from the traffic of the monitored network. While originally designed for optimizing traffic handling and accounting in the network, flow data appeared to be powerful for intrusion detection. NBA Systems based on machine learning techniques are able to evaluate these data and to recognize anomalies in the network. However, these systems suffer from a long-lasting learning phase and are susceptible to manipulations during that time. To overcome these shortcomings, we are introducing a fast-learning modular neural network based on pre-processed components.

4.2 An Overview of IP Flow-based Intrusion Detection

Nowadays hackers are continuously attacking networked systems; in fact, it would be interesting to investigate if there are still Internet users who have not been victim of an attack yet. Considering the damage caused by the attacks (billions of U.S. dollars) [4], it is important to detect attacks as soon as possible, and take, if feasible, appropriate actions to stop them. This task is particularly challenging due to the diversity in form (information gathering, password stealing, viruses, Trojan horses, Denial of Service (DoS)...) attacks exhibit.

For the detection of network attacks, special systems have been developed; these systems are called Network Intrusion Detection Systems (NIDS). In an attempt to find known attacks or unusual behavior, these systems traditionally inspect the contents (payload) of every packet [5, 6]. The problem of packet inspection, however, is that it is hard, or even impossible, to perform it at the speed of multiple Gigabits per second (Gbps) [7, 8]. For high-speed lines, it is therefore important to investigate alternatives to packet inspection. One option that currently attracts the attention of researchers and operators is *flow-based* intrusion detection. With such approach, the communication patterns within the network are analyzed, instead of the contents of individual packets. Research in this field is still relatively in its beginning, even if initial ideas to abstract from communication details and analyze *source/destination* pairs instead can already be found in papers published in the early 1990s (see for example Heberlein *et al.* [9] and Staniford-Chen *et al.* [10]). Nowadays special measurement systems are able to provide, for every pair of IP addresses and port numbers, aggregated information, such as the time data exchange has started, the time it has stopped, the amount of transferred bytes and the number of sent packets. These systems export this information in the form of Netflow [11, 12] or IPFIX [13] records to systems that analyze them. These analysis systems can then be used to detect intrusions.

In our opinion, flow-based detection can be seen as a complement of packet inspection, and should not be seen as a replacement. Both approaches can be combined into a two-stage detection process. At the first stage, flow-based approaches can be used to detect certain attacks. At the second stage, packet inspection can be used to additionally protect critical servers or selected systems, for which the first stage has discovered suspicious activities.

This section provides a survey of current research in the area of flow-based intrusion detection. This means that we consider only contributions in network intrusion detection that make explicit use of network flows as their main input. To limit our scope, we will not consider payload-based methods; readers interested in such methods can refer to existing literature [14, 15, 16, 17, 18]. Since we concentrate on network flows, we do not consider host-based intrusion detection systems. Last, since details of commercial products are hard to obtain, these have also been left out of this survey.

The section is organized as follows: Section 4.2.1 describes the motivations that have encouraged researchers to start this research. Section 4.2.2 explains the concept and ideas behind flows, as well as the network infrastructure needed for flow monitoring and analysis, such as intrusion detection. Section 4.2.3 provides a classification of current attack techniques, whereas Section 4.2.4 provides a classification of defense techniques.

Section 4.2.5 discusses how flow information can be used to detect intrusions; in this section, the focus is on thwarting Denial of Service (DoS), scans, worms and Botnets. Finally, Section 4.2.6 presents some conclusions and discusses the strengths and weaknesses of current flow-based approaches.

4.2.1 Motivation

The Internet is a complex system in constant evolution. Nevertheless, it is possible to make some observations with respect to security.

A first observation is that the number of attacks continues to grow. The Cert Coordination Center [19], one of the most well-known risks, security threats and incidents response centers, offers summaries of the yearly security situation of the Internet. The Cert/CC maintains a database of vulnerabilities, with the aim to categorize them according to their severity level and damaging impact on the systems. Vendors, system administrators and users are encouraged to submit vulnerabilities. In a similar way, in the past, Cert/CC asked the Internet community for collaboration in order to report the incidents the users were subject to. Cert/CC defines an incident as *the act of violating an explicit or implied security policy* [19]. This definition, according to the Cert/CC, covers attempts to gain access to (information on) a system, Denial of Service, disruptions, unauthorized uses and changes to hardware and software.

Since 1995, Cert/CC published each year the number of catalogued vulnerabilities. In fact, the reporting of incidents started already in 1988, but ended in 2003. The reason to stop can easily be understood from Figure 6: the growth of reported incidents is nearly exponential, while the number of catalogued vulnerabilities shows a slower growth factor. The Cert/CC itself [19] gives the following explanation:

> "Given the widespread use of automated attack tools, attacks against Internet-connected systems have become so commonplace that counts of the number of incidents reported provide little information with regard to assessing the scope and impact of attacks. Therefore, we stopped providing this statistic at the end of 2003."

A second observation is that Internet traffic, as well as line speed, continues to grow. Nowadays an access speed of 1-10Gbps is not unusual. A university network, for example, reaches traffic averages in the order of hundreds of Mbps, with high activity peaks in the order of Gbps. On backbone networks, the throughput will even be higher. Internet2 [20], for example, publishes weekly reports of the Abilene traffic. Figure 7 shows the growths in the period 2002-2008.

It is clear that Network Intrusion Detection Systems should be able to handle the growing number of attacks, the growth in Internet traffic as well as the increase in line speed. Researchers assess the current, payload-based, NIDS processing capability to lie between 100Mbps and 200Mbps [7, 8]. Well known systems like Snort [5] and Bro [6], exhibit high resource consumption when confronted with the overwhelming amount of data found in today's high-speed networks [21]. In addition, the spread of encrypted protocols poses a new challenge to payload-based systems. An example is the work of Taleb *et al.* [22, 23],

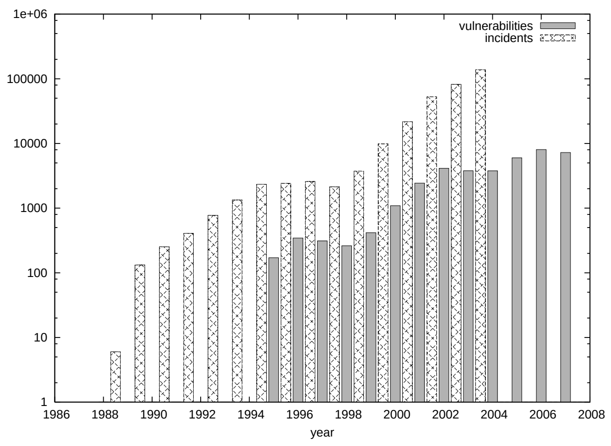


Figure 6: Trends in incidents and vulnerabilities (logarithmic scale).

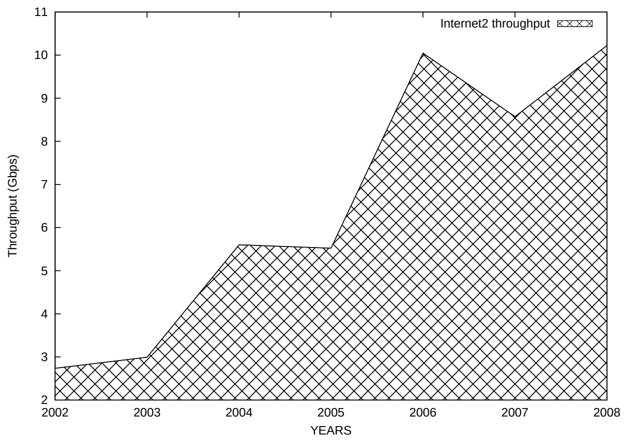


Figure 7: Network throughput (Gbps) for the network Abilene [20].

where the authors propose an intrusion detection systems based on per-packet inspection that rely only on header information in order to identify misuses in encrypted protocols.

Given these problems, flow based approaches seem to be a promising candidate for Intrusion Detection research.

Flows are created by specialized accounting modules usually placed in network routers. The same modules are responsible of exporting the flows to external collectors (see Section 4.2.2). Flow-based Intrusion Detection Systems will analyze these flows and detect attacks. Compared to traditional NIDS, flow-based NIDS have to handle considerable lower amount of data. For example, in the case of the University of Twente network, we calculated that the ratio between packets exported by NetFlow (containing the flow records) and the packets on the network is in average equal to 0.1%. Moreover, considering the network load measured in bytes, the overhead due to Netflow is in average 0.2%. Flow based intrusion detection is therefore the logical choice for high-speed networks. However, there might exist situations in which the benefit of using flows is not so pronounced. The worst case scenario would be when a flow is created for each packet passing through the monitoring point, as a consequence of a distributed DoS attack (DDoS), for example. In this case, the number of flows would increase dramatically and extra load would be put on the monitoring and analysis systems. To mitigate this problem, or, in general, to

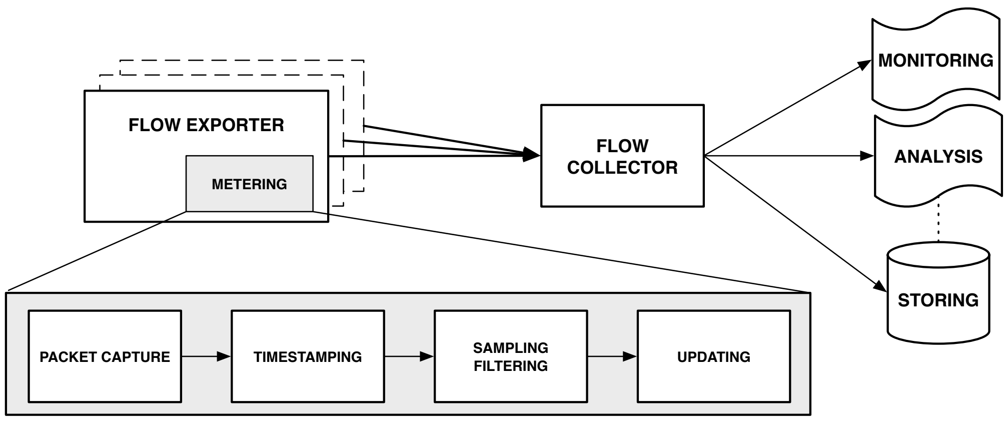


Figure 8: IP Flow exporting and collecting architecture [25, 11].

improve the performance of routers and monitoring stations, sampling techniques or flow aggregations [24] can be applied.

Sometimes it is argued that flows do not carry enough information, compared to payload inspection, for being useful for intrusion detection. The answer to this question highly depends on the user's goals. Flows, which represent by nature aggregated information, do not carry any payload. They, therefore, do not provide the detection precision of packet-based inspection, which allows for example *pattern matching* in payload content. Flows are limited to information regarding network interactions. With this information, it is still possible, however, to identify communication patterns between hosts, when communication takes place and which amounts of packets and bytes have been moved. For many attacks, this information is sufficient. In any case, it is important to underline that flow-based intrusion detection is not supposed to substitute the packet-based one, but rather complements the approach by allowing early detection in environments in which payload-based inspection is not feasible. As described by Schaffrath *et al.* [26], in an ideal world payload-based solutions would always outperform flow-based ones in accuracy. In high-speed networks, however, the processing capabilities of the NIDS may be too limited to allow payload-based approaches.

4.2.2 IP Flows

In the last decade, flows have become quite popular in IP networks. Nowadays all major vendors equip their routers with flow accounting capabilities. Traffic information is collected and stored in flow records that provide an overview of network usage at different levels of granularity.

Flow definition

In literature, several definitions of an IP flow can be found [11, 27, 12]. This article follows the definition of *IP flow* as it was described by the IPFIX (IP Flow Information Export) working group within IETF [13, 25]:

> "A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties."

In the IPFIX terminology, the *common properties* are called *flow keys*: they are, for example, source and destination addresses, source and destination port numbers and IP protocol:

$$(\mathtt{ip\_src}, \mathtt{ip\_dst}, \mathtt{port\_src}, \mathtt{port\_dst}, \mathtt{proto}).$$

Aggregated views on the network traffic can be obtained by choosing coarser grained flow definitions, according to the need of the network administrator, as discussed in the work of Fioreze *et al.* [27]. It is important to underline the difference between *flows* and *connections*, as used in the case of TCP. A flow can exist also in situations in which there is no TCP connection: an example of this is a UDP flow, where a set of packets has been sent from a certain source address/port to a certain destination address/port. Moreover, a flow does not have size restrictions: each communication between source and destination hosts will generate a flow, even if a single packet has been exchanged.

Accounting flows is a two-step process: *flow exporting*, and *flow collection*. These tasks are performed by two components: *flow exporter* and *flow collector*. Figure 8 shows this exporting/collecting process.

The *flow exporter*, also known as *observation point*, is responsible for the *metering* process, i.e., creating flow records from observed traffic. The *flow exporter* extracts the packet header from each packet seen on the monitored interface. Each packet header is marked with the *timestamp* when the header was captured. After that the header is processed by a *sampling-filtering* module, where it can be *sampled* or *filtered*. The final step is the *update* module. Each incoming packet header triggers an update to a flow entry in the *flow cache*. If there is no flow matching the packet header, a new flow entry is created. Once a flow record *expires*, it is sent to the flow collector. In case of Cisco NetFlow [11] and similarly in IPFIX [28], a flow is considered expired when:

- the flow was idle (no packets have been detected in the flow) for a longer time than a given threshold (known as *inactive timeout*). The default value for the inactive timeout for Cisco Netflow [11] is, for example, 15 seconds, but it can be changed according to the requirements of the network to be monitored.
- the flow reaches the maximum allowed lifetime. When this happens, its corresponding flow record is exported to the collector and, if necessary, a new flow record is created for that flow (*active timeout*). For Cisco Netflow, the active timeout is 30 minutes, but our experiences showed that shorter timeouts are also common. At the University of Twente, for example, an active timeout of 1 minute is used.
- the `FIN` or `RST` flags have been seen in a TCP flow.
- the flow-cache memory gets full. In this case, certain flow records are marked as expired and exported to the collector. Least Recently Used (LRU) algorithms may be used to free the flow-cache memory, as well as heuristic algorithms.

The aim of the *flow collector* is to retrieve the flows created by the flow exporter and to store them in a form suitable for further monitoring or analysis.

Flow Export Protocols

A flow export protocol defines how flow records are transported between an exporter and a collector. Netflow version 5 [11], developed by Cisco, has a very simple flow export protocol that transports flow records of fixed size (48 bytes in total). A Netflow v5 flow record contains source and destination IP addresses and ports, start and end timestamps, type of service, level 3 protocol, TCP flags next hop router, input and output SNMP interfaces, source and destination autonomous systems and network masks. Moreover, each flow carries aggregated information about the amount of packets and bytes exchanged.

Netflow version 9 and IPFIX propose flexible protocols in which flow record formats can be defined by using templates. The latter protocols allow also a larger set of parameters to be used as flow keys. An IPFIX packet is logically divided into sections known as *sets*. A message can normally consist of three kinds of sets, namely *Template sets* (format template exchange), *Data sets* (flow records) and *Options Template Sets* (necessary for the correct interpretation of a Template set). For a more detailed treatment of the IPFIX message format, see [25].

Sampling

IP flow accounting requires state information to be kept for each active flow. On high-speed links, there may be millions of packets per second and hundreds of thousands of active flows. If for each incoming packet, a flow lookup is performed and state information is kept for each flow, a heavy demand will be put on the CPU and memory resources of the flow exporter. In order to reduce this demand, sampling methods can be deployed. The IETF PSAMP (Packet Sampling) working group [29] is currently discussing the creation of possible standards in this area. It should be noted, however, that sampling not only lowers the demands put on the flow exporter, but also makes detection of intrusions harder. Several studies discuss the impact of sampling on intrusion detection and flow accounting. Examples are Brauckhoff *et al.* [30], Mai *et al.* [31] and Zseby *et al.* [32].

Two main categories of sampling can be identified: packet sampling and flow sampling.

- **Packet Sampling**: as explained in Izkue *et al.* [33], Wang *et al.* [34] and He *et al.* [35], sampling techniques can be divided into *systematic* and *random* ones. In *systematic* packet sampling, a packet is deterministically selected on the base of a time interval (*time-driven sampling*) or a sequence of packet arrivals (*event-driven sampling*). For example, it is possible to select a packet every t seconds, or a packet every n packets. In *random* packet sampling, on the other hand, the sampling process relies on a probability distribution function. The two main classes of random sampling are:

  - *n-in-N sampling*: The traffic is split into sequences of *N* packets. Out of these, *n* are randomly selected.
  - *probabilistic sampling*: Each packet is sampled with probability p. This sampling probability p can be fixed, or can depend on specific packet characteristics, such as for example the packet size.

  The deployment of one or more packet sampling strategies depends on which traffic characteristic the administrator is interested in. NetFlow is using an *n-in-N* sampling technique, usually in the form of *1-in-N* sampling.

- **Flow sampling**: similarly to random packet sampling, random flow sampling algorithms sample each flow with a random probability. *Sample and hold*, for example, is a sampling method proposed by Estan *et al.* [36] that accurately accounts for large flows. In this case, when the system detects the presence of a new packet that does not belong to any already existent flow, it creates a flow entry with probability p. If the new flow is created, all following packets belonging to the flow will be accounted, as opposed to packet sampling in which each packet independently undergoes the sampling procedure. It is easy at this point to imagine why this sampling strategy is biased towards large flows. Duffield *et al.* [37, 38] and Alon *et al.* [39] proposed *Smart Sampling* as a method to dynamically control the size of sampled data. *Smart Sampling*, both in the form of *threshold sampling* [38] and *priority sampling* [39], is based on the observation that packets and bytes in flows follow a heavy tailed distribution. A simple flow sampling strategy may omit flows that have large impact on the estimation of the total traffic of the network. To overcome this problem, Duffield *et al.* and Alon *et al.* propose sampling schemes in which the probability that a flow will be sampled depends on its size.

This section gave an overview of how flows are created. To understand how flows can be used for intrusion detection, we are now going to give a brief overview of the attacks present in our networks.

4.2.3 Attack Classification

Several attack classifications have been described in literature [40]. These classifications usually distinguish between the following basic categories [41, 42]:

- **Physical attacks:** attacks based on damaging the computer and network hardware.
- **Buffer overflows:** attacks that gain control or crash a process on the target system by overflowing a buffer of that process.
- **Password attacks:** attacks trying to gain passwords, keys, etc. for a protected system.
- **(Distributed) Denial of Service attacks:** an attack which leads to situations in which legitimate users experience a diminished level of service or cannot access a service at all.
- **Information gathering attacks:** an attack that does not directly damage the target system, but gains information about the system, possibly to be used for further attacks in the future. This category comprises network traffic sniffing and (port) **scans**.
- **Trojan horses:** a program disguised as a useful application, which deliberately performs unwanted actions.
- **Worms:** a program that self-propagates across a network. Self-propagation is the characteristic that differentiates worms from viruses (see below). A worm spread can be extremely fast: an example is the Sapphire/Slammer worm, which is known to have infected 90% of the vulnerable hosts in 10 minutes [43].

- **Viruses:** a virus is regarded as a worm that only replicates on the (infected) host computer. Hence, it needs user interactions to propagate to other hosts. Often, the definition also requires that a virus has to attach itself to files on the host, e.g., executable files, in order to be activated. As a consequence, the speed of spreading cannot be compared with a worm spread.

In addition, Hansman *et al.* [42] summaries under the category “Network attacks” various other attacks, such as spoofing, session hijacking and parameter tampering.

The previous categories should not be regarded as mutual exclusive classes of attacks. For example, buffer overflows and port scans can be regarded as separate categories of attacks, but also as specific techniques used by worms and DoS attacks. Rather, these categories describe general “concepts” of attacks that have been frequently observed in practice. Note that not all taxonomies provide a classification like the one given above. For example, Howard [44] focuses on a process-driven taxonomy, based on the objective of the attacker, the used tools, etc.

Nowadays, an additional threat has evolved pertaining Botnets. **Botnets** are groups of computers “infected with malicious program(s) that cause them to operate against the owners’ intentions and without their knowledge”, as defined in Lee *et al.* [45]. Botnets are remotely controlled by one or more *bot-masters*. Moreover, Botnets are the perfect infrastructure for setting up and supporting any kind of distributed attack, such as, for example, DoS attacks and SPAM campaigns. Infected hosts unknowingly become part of Botnets, and take part in malicious activities [46, 47]. The threats posed by Botnets are such that we decided to include them in our attack classification.

Flow-based intrusion detection, since it relies only on header information, can address only a subset of the attacks presented above. In particular, the research community currently provides approaches to detect the following classes of attacks:

- Denial of Service;
- Scans;
- Worms;
- Botnets.

Approaches to detect these attacks will be further discussed in Section 4.2.5.

4.2.4 Detection classification

According to Halme *et al.* [48], an Intrusion Detection System is an *anti-intrusion approach* that aims to *discriminate intrusion attempts and intrusion preparation from normal system usage*. Since the first papers on intrusion detection appeared in the Eighties of the previous century, several taxonomies of intrusion detection techniques were proposed. Our study identifies two main contributions to the field, the work of Debar *et al.* [14, 15] and that of Axelsson [16].

Debar *et al.* [14, 15] were among the first to propose an intrusion detection system taxonomy. Their classification focuses on the following elements:

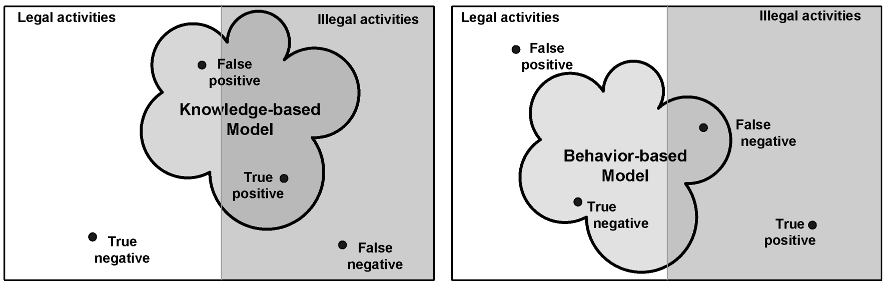


Figure 9: Detection capabilities of different intrusion detection models [49].

- **Detection Method**: if a system bases the detection on a definition of *normal* behavior of the target system, it is called *behavior-based*. If it matches the input data against a definition of an attack, it is known as *knowledge-based*. In literature, the community usually refers to these classes with the names of *anomaly-based* and *misuse-based* solutions [49, 16, 17, 50, 51].
- **Behavior on detection**: a system can be proactive and act against the intruder (*active system*) or can generate alerts that will be later processed by a different system or a human operator (*passive system*).
- **Audit source location**: the data processed in order to detect intrusion can be *host* or *application logs*, *network packets* or *alerts* generated by other detection systems.
- **Detection Paradigm**: the IDS can detect the current status of the target system (secure or insecure) or can alert on a state transition (from secure to insecure).
- **Usage frequency**: the system can perform its task in real-time (*continuous monitoring*) or post-mortem (*periodic analysis*)

Axelsson [16] bases his taxonomy on the one proposed in Debar *et al.* [14, 15], but extends and completes it. In particular, beside the previously described characteristics, a system is described also on the basis of the following:

- **Locus of data-processing**: a system can be *centralized* or *distributed*, irrespectively of the origin of the data.
- **Locus of data-collection**: the data collection can be *centralised* or *distributed*.
- **Security**: the intrusion detection system can be itself target of security threats.
- **Degree of inter-operability**: a system can be built to work in *conjunction* with other systems (exchanging data) or *stand-alone*.

In his work, later followed by Almgren *et al.* [52], Axelsson focuses on detection methods, once again divided in two classes: anomaly-based and misuse-based. In that work, an *anomaly-based* system can be described as:

- **Self-learning**: the system is able to automatically build a model of the normal behavior of the system, or:

- **Programmed**: the definition of normality has to be provided by the system developer.

A *misuse-based* system, on the other hand, presents a unique subclass, *programmed*: the system is provided with a knowledge-base of attacks, against which it matches the inputs.

Figure 9 shows the detection capabilities of legal and illegal activities, for misuse (knowledge-based) and anomaly (behavior-based) systems, respectively. A misuse-based model is supposed to describe only illegal activities. In some cases, however, if the system is not *accurate* enough, legal activities can be flagged as intrusions; such events are called *false positives*. At the same time, if the model is not *complete*, it will not be able to report all malicious activities; unflagged illegal activities are known as *false negatives*. An anomaly-based model, on the other hand, is supposed to describe only legal activities (*normality*). Also in this case, incompleteness and inaccuracy can lead to false positive and false negatives.

Finally, in [16], Axelsson introduces a third class of systems in which both anomaly-based inspired characteristics and misused-based ones coexist. In his work, such systems are known as *compounds*.

In their taxonomies, Debar *et al.* [14, 15] and Axelsson [16] consider a wider spectrum of categories, including some that are outside the scope of this paper. For example, they distinguish between *host-based* and *network-based* detection approaches. The former analyses the status of a single host, monitoring internal functionality (e.g., CPU usage, system call traces, log-in attempts). The latter bases its analysis on network information and can monitor an entire network. In this paper, we are only interested in this second kind of systems, since we are analyzing network data only. In particular, we are interested in flow-based approaches; since such approaches are relatively new, the taxonomies found in literature do not explicitly consider them. Despite this, since flows represent network-based aggregated data, we still consider the taxonomies useful for our purpose.

4.2.5 Flow-based solutions

This section presents the state of the art solution for each category of attack that can be detected using flows (see Section 4.2.3). Moreover, it classifies each contribution according to the taxonomies presented in Section 4.2.4.

Denial of Service

Detection of Denial of Service is often addressed in flow based intrusion detection. These attacks, by their nature, can produce variations in the traffic volume that are usually still visible at flow scale.

It is important to underline, nevertheless, that in case of flow-based detection we are implicitly addressing the problem of *brute force* DoS attacks, i.e., a type of DoS that relies on resource exhaustion or network overloading. Unfortunately, it is almost impossible to directly detect *semantic DoS* attacks, i.e., attacks in which the service interruption is caused by the payload contents. For example, let us consider the (nowadays out-of-date) Ping of Death attack. In such attack, the attacker sends malformed or otherwise malicious ping packets, which causes the victim system to crash. Since this attack does generate a single ICMP flow, the attack would most likely go undetected. There would be, indeed, no change in flow frequency and intensity. A different case of semantic attack would be one that changes the distribution of flows. An example is the DoS effect related to the scanning phase of the Sapphire/Slammer worm [43]: while spreading, the worm provokes the crash of Microsoft SQL Server hosted on the target machine. Nevertheless, at flow basis, the detection of this attack would be most probably related to the scanning phase, and not to the DoS itself.

An overview of how often DoS attacks appear in practice is given in Moore *et al.* [53]. They estimate that, based on an analysis conducted over multiple one-week traces for a period of three years, on average the number of different victim IPs on the entire Internet is 24.5/hours. Even though Moore *et al.* do not specify if they investigated brute force or semantic attacks, the statistics clearly show that DoS attacks detection is, still in these days, a problem that requires experts' attentions.

There are two main examples of anomaly-based DoS detection in high-speed networks, using flow information only. The work of Li *et al.* [54] and Gao *et al.* [55], in the first place, approach the problem using aggregate flow measures collected in appropriate data structures, named *sketches*. A sketch is originally a one-dimensional hash table suitable for fast storing of information: it mainly counts occurrences of an event. In their papers, the authors work with 2D sketches, a more powerful extension of the original ones, in which, for each dimension, a set of flow-derived fields is hashed. Sketches permit to statistically characterize how the traffic varies over time, simply by tracking the presence of a flow in a specified time frame. An anomaly-based engine triggers alarms based on a forecast value of the measure the system is supposed to monitor: a sharp variation from the mean is flagged as an anomaly. A simple example of the use of sketches in DoS attacks is the detection of SYN Flooding attacks [56], as described in Gao *et al.* [55]. In this case, the sketch is supposed to store, for each time frame and each tuple `(dest_IP, dest_port)`, the difference between the number of SYN packets and the number of SYN/ACKs. If the stored value for the current time deviates from the expected one, a DoS SYN Flooding attack is going on. The sketch-based approach could potentially be deployed also without the use of flows, relying in this case on header inspection. Nevertheless, in this case the data reduction gain provided by flows would be most probably lost. Gao *et al.* developed a prototype that receives exported flows from a netflow-enable router in real time.

A similar approach is proposed by Zhao *et al.* [57]. In this case, a data-streaming algorithm is used to filter part of the traffic, and identify IPs that show an abnormal number of connections. The authors consider both the case in which a host is the source of an abnormal number of outgoing connections (*large fan-out*), as well as the case in which a host is the destination of an unusual number of connection attempts (*large fan-in*). The first case matches the definition of a scanning host, while the second is used for detecting DoS victims. The method is based on 2D hash tables, clearly resembling the work of [54] and [55]. In their paper, Zhao *et al.* also apply a flow *sampling* algorithm (see Section 4.2.2), to reduce the amount of data to be processed and significantly raise processing speed. At the same time, since sampling further reduces the available information, the authors developed statistical formulas to accurately estimate the *fan-in/fan-out* of the considered hosts.

A more detailed approach is presented by Kim *et al.* [58]: in this paper many different DoS attacks are described in terms of traffic patterns, based on flow characteristics. In particular, the authors focus on the number of flows and packets, the flow and packet sizes, total bandwidth used as well as average flow size and number of packets per flow. An example of attack pattern is the one produced by a SYN Flooding attack: a large flow count, yet small packet counts, as well as small flow and packet sizes and no constraints on the bandwidth and the total amount of packets. The pattern is significantly different from the one generated by an ICMP or UDP flooding attack, in which we have large bandwidth consumption and the transfer of a large number of packets. Kim *et al.* clearly identify the metrics they are interested in and formalize them into *detection functions* that give the likelihood of a traffic pattern representing an attack.

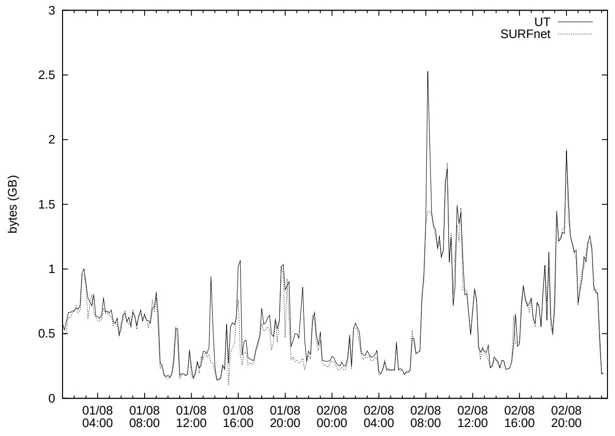
In the context of DoS monitoring and detection, it is important to cite also the work of Münz *et al.* [59], which propose a general platform for DoS detection. The system, known as TOPAS (Traffic flOw and Packet Analysis System), acts as a flow collector for multiple sources and locations, offering preprocessing capabilities in order to obtain an information format suitable for further processing. On this platform, many different detection modules can run in real-time according to the necessities of the network administrator. Examples of modules are a *SYN flood detection module*, a *traceback module* (to allow identification of the entry point of spoofed packets in the attacked network) and a *Web Server overloading module* (focusing on DoS attacks using HTTP requests). The work has been developed within the context of the European Diadem Firewall project, which specifically focuses on DoS and DDoS detection [60].

Attention must also be given to the work of Lakhina *et al.* [61, 62, 63, 64]. The analysis is conducted on flow aggregation, namely on origin-destination flows between Points of Presence (PoP) on the Abilene [20] and Sprint-Europe [65] networks. On this small set of pairs (only n^2, where n is the number of PoPs), it is possible, through principal component analysis, to decompose the traffic flowing through the backbone in time related traffic trends (*eigenflows*). There are three types of eigenflows: deterministic eigenflows that show a periodical trend (day-night pattern), spike eigenflows that show isolated values that strongly deviate from the average and noise eigenflows that appears to be roughly Gaussian. The spike components reveal the presence of a traffic anomaly. The proposed method is general enough to capture various kinds of anomalies, due to failures or attacks, and is appropriate for almost all the attack classes we are interested in (DoS, scans and worms).

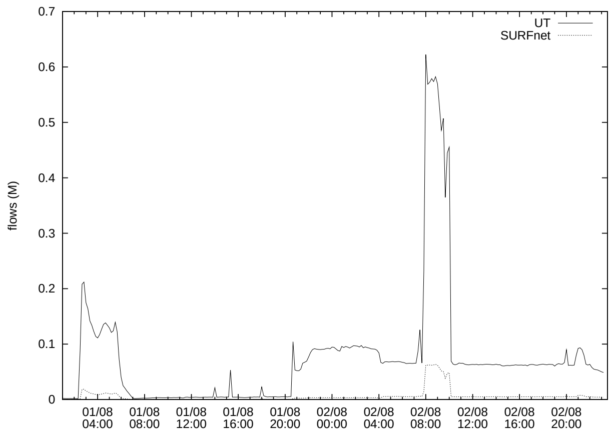
Scans

Scans are usually characterized by small packets that probe the target systems. Keeping this characteristic in mind, it is easy to imagine that scans can easily create a large number of different flows. There are three categories of scans: (i) a host scanning a specific port on many destination hosts (*horizontal scan*); (ii) a host scanning several ports on a single destination host (*vertical scan*); (iii) a combination of both (*block scan*). Irrespectively of the kind of scan, the result will be a variation of the flow traffic in the network. At the same time, scans are less likely to have impact on the total traffic volume, as shown in Sperotto *et al.* [66].

Figure 10 shows an example of SSH flows captured in 2007 at the University of Twente (UT) and SURFnet [67], the UT Internet service provider. The byte time series (Fig. 10(a)) is quite irregular, with sharp high- and down-peaks that do not clearly indicate the presence of an attack. On the other hand, the flow time series (Fig. 10(b)) shows sudden and



(a) Bytes



(b) Flows

Figure 10: Byte (a) and flow (b) time series for SSH traffic at the University of Twente network and SURFnet [66].

frequent peaks, during which the number of flows can rise to several hundreds of thousands per observation bin (10 minutes, in the case of Figure 10). After a more detailed analysis, these peaks appeared to correspond to multiple SSH scanning sessions, trying to guess user names and passwords. An interesting difference between SURFnet and the UT is that SURFnet applies 1:100 packet sampling, whereas the UT does not apply packet sampling. Still Figure 10 shows that scans can even be detected in SURFnet, despite the sampling.

In literature, scans have generally been investigated by considering their most obvious characteristic: the scanning source shows an unnaturally high number of outgoing connections. The problem has been approached in this way by Zhao *et al.* [57], already cited in the previous section. Looking at host behavior from an incoming/outgoing connection perspective allows addressing DoS and scan attacks as faces of the same problem: hosts with an inadequate and unusual fan-in/out. Similarly, Kim *et al.* [58] attempt to describe a scan in terms of traffic patterns, as already explained in the case of DoS. The authors differentiate between network (horizontal) scans and host (vertical) scans.

The approach described in Wagner *et al.* [68] is not related to traffic volume anomalies. In this case, the probabilistic measure of entropy is used to disclose regularity in connection-based traffic (flows). Entropy has been introduced in Information Theory in 1948 [69] and, generally speaking, is a measure of randomness and *uncertainty* of a stochastic process. Entropy is also related to lossless data compression: the theoretical limit of the compression rate of a sequence of bits is exactly the entropy of the sequence. Starting from this well-known result, Wagner *et al.* created an efficient analysis procedure based on compression of sequences of network measurements. They observe that, in the case of a scanning host, the overall entropy in a specific time window is subdued to a change. In particular, the presence of many flows with the same source IPs (the scanning host) will lead to an abrupt decrease of the entropy in the distribution of the source IP addresses. At the same time, the scanning host will attempt to contact many different destination IPs on (possibly) different ports, generating an increase in these entropy measurements. The combined observation of multiple entropy variations helps in validating the presence of an attack. Other approaches are based on logistic regression [70] and distances from baseline models [71].

Worms

Worm behavior is usually divided into a target discovery phase (the worm explores the network in order to find vulnerable systems) and a transfer phase (the actual code transfer takes place) [74, 75]. Code Red [76] and Sapphire/Slammer [43] are examples of this mechanism. Flow-based detection systems usually focus on the target discovery phase, since the transfer of malicious code cannot easily be detected without analyzing the payload. In many cases, worm detection can be similar to scan detection, and many researchers use the same approach for both threats. The approach adopted by Wagner *et al.* [68], for example, can naturally be extended to worms, as well as the ones of Zhao *et al.* [57] and Gao *et al.* [55].

Dübendorfer *et al.* [72] and Wagner *et al.* [73] attempt to characterize the host behavior on the basis of incoming and outgoing connections. The proposed algorithm assigns hosts to a set of classes. The definition of these classes is such that only suspicious hosts will belong to them. The *traffic class* groups hosts that send more traffic than what

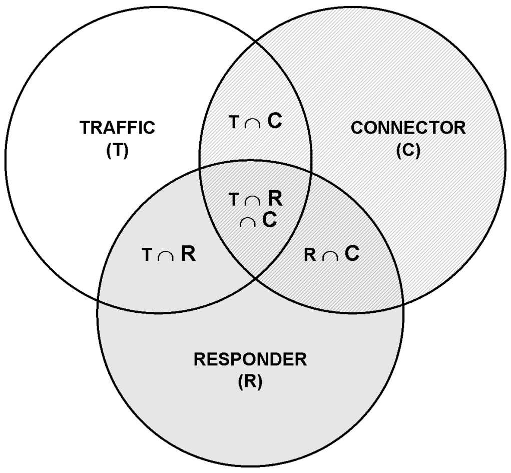


Figure 11: Host classes and their intersections[72, 73].

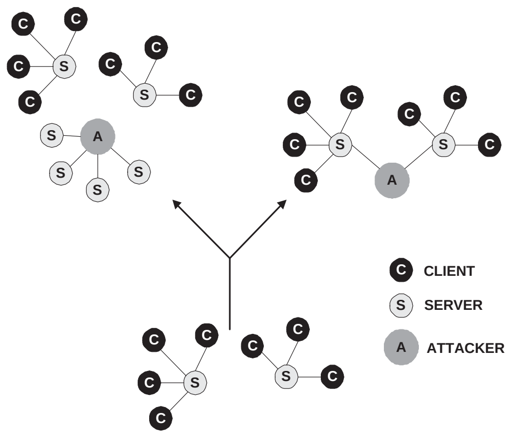


Figure 12: Example of graph based hit-list worm spreading analysis [78].

they receive. Hosts that show an unusual high number of outgoing connections are part of the *connector class*. Finally, hosts involved in many bidirectional connections belong to the *responder class*. In the proposed model, a host can belong to more than one class. Figure 11 describes the three classes and their possible intersections. The method aims to periodically check the status of the hosts of an entire network. In this way, it is able to detect worm spreads, as they cause massive changes in the cardinality of one or more classes. Moreover, Dübendorfer *et al.* [72], by properly filtering the interesting flows, manage to identify both e-mail spreading worms and scanning worms, without concerns about their scanning strategies.

A different approach is taken by Dressler *et al.* [77], that uses the correlation between flows and honeypots logs. In this case, the need for a *ground truth*, i.e., a trusted source of information for the system validation, made the authors rely on a honeypot. In this way, deploying at the same time honeypot, flow monitor and a collecting database, it is possible to carefully identify *worm flow-signatures*, that is sequence of connections and flow-related information about the scanning and transmitting behavior of a worm. According to the presented results, the approach seems to be promising.

Finally, Collins *et al.* [78] propose a solution to the problem of *hit-list worms* detection. A hit-list worm is a worm that bases its scanning strategy on the sequential probing of

Table 1: Categorization of the proposed solutions according to the taxonomy.

System	Detection Method	Behaviour on detection	Usage Frequency	Data processing	Data collection
Li *et al.* [54] Gao *et al.* [55]	anomaly	active	real-time	centralised	distributed
Zhao *et al.* [57]	not spec	not spec	real-time	centralized	distributed
Kim *et al.* [58]	misuse	passive	real-time	centralized	distributed
Münz *et al.* [59]	compound	passive	real-time	centralized	distributed
Lakhina *et al.* [61, 62, 63, 64]	anomaly	passive	real-time	centralized	centralized
Wagner *et al.* [68]	anomaly	passive	real-time/batch	centralized	centralized
Gates *et al.* [70]	misuse	passive	batch	centralized	centralized
Stoecklin *et al.* [71]	anomaly	passive	batch	centralized	centralized
Dübendorfer *et al.* [72] [73]	compound	passive	real-time	centralized	centralized
Collins *et al.* [78]	anomaly	passive	real-time	centralized	centralized
Dressler *et al.* [77]	misuse	passive	real-time	centralized	centralized
Karasaridis *et al.* [79]	misuse	passive	real-time	centralized	centralized
Livadas *et al.* [47] [80]	not spec	passive	batch	centralized	centralized
Gu *et al.* [81]	anomaly	passive	real-time	centralized	centralized

a predefined list of hosts that are supposed to be always online. This technique is used because worms usually have a slow initial spreading phase, and the use of a hit-list consistently increase the initial infection speed. Since hit-lists are commonly used to start infections, detecting them as soon as possible may be quite useful. Collins *et al.* employ a graph-based algorithm that slices the network according to a monitored protocol (like HTTP, FTP, SMTP, or Oracle). They argue that the number of hosts normally involved in the use of a certain protocol, i.e., the number of vertexes in the graph, is in average regular over time. Also the pattern of communication between hosts, i.e., the cardinality of the connected components in the graph (connected subgraphs with a maximal number of vertexes), has the same property. This regularity is disturbed only when a new host starts to scan the network following a hit-list: in this case, the authors observe a larger number of vertexes in the graph (the scanned hosts) and a drastically enlarged cardinality of the connected components. The scanning host, indeed, will communicate with servers (in its hit-list) that in normal conditions do not have any connection. Figure 12 shows cases of hit-list infections that modify the number of vertexes in the graph and the cardinality of the largest connected component. In the example, two servers, depicted at the bottom of the figure, communicate with, respectively, four and three clients during a normal observation period. The two disjoint sets of hosts will form two connected components in the *protocol graph*. During a malicious observation period, i.e., while a hit-list worm is spreading, two situations can be observed. In the first one, on the upper left of the figure, the attacker contacts servers that do not normally appear in the monitored traffic: as consequence, the cardinality of the vertex set in the protocol graph will increase, meeting the first detection condition in [78]. In the second case, on the upper right corner, an attacker will contact both servers, since they are on its hit-list. As a consequence, the two connected components described in the example will be reduced to one, meeting the second detection condition.

Botnets

As explained in Section 4.2.3, Botnets consist of infected hosts (bots) controlled by a central entity, known as master (or bot-master). As these networks tend to be spread over multiple administrative zones, complete identification of bots is a difficult problem.

Since bots are no longer harmful once the master is isolated, a straightforward mitigation approach is to identify the master. Nevertheless, as Zhu *et al.* [82] pointed out in their survey on Botnet research, the defense against botnets is not yet efficient and the research in this field is still in its infancy.

As a fact, many Botnets used to rely on IRC channels, which can be identified at flow level, as described in the work of Karasaridis *et al.* [79]. The authors propose a model of IRC traffic that does not rely on specific port numbers. Karasaridis *et al.* address two main points. First, they propose a multistage procedure for detecting Botnets controllers. Starting from reports of malicious activity obtained from diverse sources (e.g., scan logs, spam logs, and viruses), the authors identify groups of flows involved in suspicious communications (*candidate controller conversations*). These conversations may happen between a host and a candidate server (*controller*) that use either an IRC port (e.g., 6667, 6668 or 7000) or that hides the control traffic using a different protocol. In the second case, the candidate conversation is checked against the flow model. The second aim of Karasaridis *et al.* is, once the controllers have been identified, to group the suspected bots into behavioral groups, i.e., clusters of bots that show the same activity pattern. For this purpose, they suggested a hierarchical clustering procedure that groups the host based on their port activities. In [79], the authors also explain why Botnet detection slightly differs from scan or DoS detection. For scans and DoS, current research aims at *real time* identification, with alerts that permit the network administrator to intervene as soon as possible. In the case of Botnets, only long time observations can lead to the identification of the bots and controller.

In a similar way, the work of Livadas *et al.* [80] and Strayer *et al.* [47] approach the problem by modeling the TCP flows of IRC chats. The authors present the first results of a study pertaining to the use of machine learning techniques for Botnet traffic identification. In particular, they structure their approach in order to answer two research questions: is it possible to distinguish between i) IRC and non-IRC traffic; ii) botnet IRC traffic and normal IRC traffic. In the paper, the effectiveness of machine learning methods, such as Naive Bayes classifiers, Bayesian networks and classification trees, is tested. The input is an enriched version of flows (including additional information, such as variance of the bytes per packet in the flow, or the number of packets for which the PUSH flag is set). The work shows that automatic identification of Botnet IRC traffic seems possible.

A different approach is proposed by Gu *et al.* [81]. They developed a Botnet detector, *BotMiner*, which is independent of Botnet Command and Control (C&C) protocols and structures. Gu *et al.* developed a detection framework that aims to characterize a Botnet according to the following definition:

> A *coordinated group* of *malware* instances that are *controlled* via C&C channels.

BotMiner sniffs the traffic at the observation point and conducts two parallel analyses. On one side, it relies on flows for detecting groups of hosts with similar communication patterns. On the other side, it inspects packet payloads (via Snort) in order to detect anomalous activities. These activities are then clustered together in order to detect groups of hosts that have similar malicious behavior. In both steps, unsupervised clustering techniques have been used. As the authors describe, both step are necessary in order to properly identify possible bots, and a *cross correlation* phase is performed in order to merge

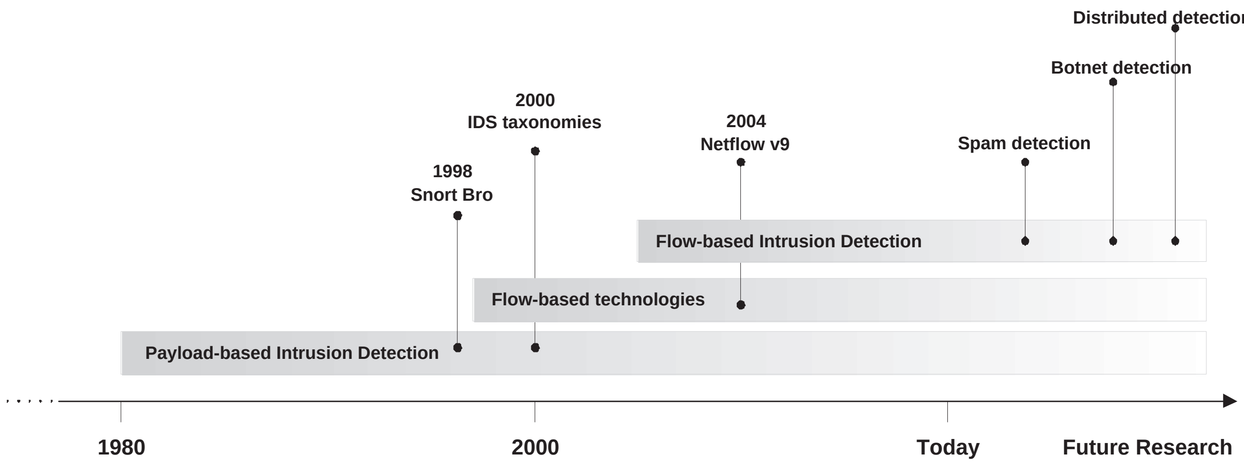


Figure 13: Time line of evolution of intrusion detection and flow-based technologies.

the results of the previous analyses and extract meaningful groups of malicious host that form a Botnet. The approach, which has already been implemented in a working prototype, shows good detection results. Moreover, it clearly shows that the problem of Botnet detection is more complex than the general problem of attack detection. A misbehaving host, indeed, is not sufficient to indicate the presence of a Botnet. More sophisticated intra-host communication analysis is needed to characterize the group nature of Botnets.

Even though Gu *et al.* [81] and Karasaridis *et al.* [79] present better results than Livadas *et al.* [80] and Strayer *et al.* [47], all the contributions clearly show that the problem of Botnet detection still remains unsolved. This is mainly due to the subtle and highly dynamic evolution of the Botnets themselves. Since the research on Botnet identification is still in its beginning phase, a strong research effort is needed to develop effective detection procedures. In this regard, flow-based approaches play an important role.

Solutions classification

The intrusion detection taxonomies presented in Section 4.2.4 allowed us to categorize the state of the art in the field of flow-based intrusion detection. However, in our specific case, not all the categories in the taxonomies are relevant to our problem. For example, network information is the only *audit data* we are interested in, so this category has been omitted from our study. The *detection paradigm* (state/transition-based) is applicable mainly to host-based solutions, and for this reason has also been discarded. We also have not considered Axelsson's *security* class. Only one of the contributions, indeed, explicitly addresses the problem of attack resilience [55]. Finally, it is important to notice that, at the moment, the main research concern is still on developing flow-based detection engines, and less effort is put on problems like *interoperability* of different instances of the IDS, or between the IDS and other network components (firewalls, routers...). In our survey, only a few contributions specifically address this subject, such as Li *et al.* [54] and Gao *et al.* [55].

Table 1, which presents our classification, gives some insight in the current research trends in flow-based intrusion detection. As it has been for payload-based solution, also in this case, the *anomaly-/misuse-based* classes play an important role: we can see contribution in both fields. Moreover, some researchers, such as Münz *et al.* [59], Dübendorfer *et al.* [72] and Wagner *et al.* [73], developed *compound* methods. This is due to the interest in joining the strengths of both anomaly and misuse-based approaches, as well as to the increasing interest in multi-purpose platforms that offer a shared base for different detection modules. The work of Gu *et al.* [81], on the other hand, is classified as anomaly-based. It indeed uses Snort only as a complementary source of data, while the entire detection engine is based on anomaly techniques. On some occasions [47, 57, 80], the detection approach is unclear or not specified. This happens when these works address more general problems than detection, and attack identification serves only as a possible application. For example, Zhao *et al.* [57] are interested in *super-sources/destinations* as a more general problem of *scan/DoS detection*. On the other hand, Strayer *et al.* [47] and Livadas *et al.* [80] do not specify if they aim at modeling either the normal behavior of IRC conversations or the anomalous one pertaining to the command-and-control streams. Therefore, it has not been possible to classify the contribution regarding this category.

By considering the behavior on detection, the focus seems to be on passive solutions, which complete their task with the rising of an alert to the network administrator. This also means that the majority of the solutions heavily rely on human intervention for attack mitigation and blocking. At the same time, nevertheless, our classification shows that there is a clear preference for *real-time solutions*, which clearly signals the need for fast responses in flow-based security.

The majority of the contributions rely on centralized *data processing*. Flows are a powerful approach to data reduction, as already pointed out in Section 4.2.1. In many cases, a stand-alone machine will have enough processing power to deal with the flow-data stream. On the other hand, flows are particularly suitable to be exported towards remote collection points, making it extremely easy to develop a system based on distributed *data collection* points. The majority of the solutions that we studied assume a single(centralized) collection point for the ease of analysis, but the authors do not explicitly exclude the possibility of distributed collection.

4.2.6 Conclusion

This paper presented a survey of the state of the art of flow-based intrusion detection, focusing on the period 2002–2008. During this time, flow-based techniques attracted the interest of researchers, especially for analysis of high-speed networks. The recent spread of 1-10Gbps technologies, and the day by day increasing network usage and load, have clearly pointed out that *scalability* is a growing problem. In this context, flow based solutions to monitor and, moreover, to detect intrusions help to solve the problem. They achieve, indeed, *data and processing time reduction*, opening the way to high-speed detection on large infrastructures.

This paper also showed, however, that in some cases the complete absence of payload should still be perceived as the main drawback of flow-based approaches. For example, the use of flow-based techniques makes it very difficult to detect so-called *semantic attacks*; attacks for which the disruptive power is in the payload, and which do not create visible flow variations (bytes, number of packets or number of flows). Nevertheless, as mentioned in Section 4.2.1, flow-based intrusion detection is not meant to substitute payload-based solutions, but to complement them in situations where technological constraints make payload-based techniques infeasible.

Figure 13 shows, in a schematic time line, the evolution of payload-based intrusion detection, flow-based technologies and flow-based intrusion detection. Payload-based solutions constituted the first effort in developing network-based intrusion detection. Nevertheless, they are, still today, a meaningful approach to security. Figure 13 also shows the rise of flow-based technologies (see Section 4.2.2). Once flow-based monitoring became an established technology, we can see how flows became also a source of data for intrusion detection (see also Section 4.2.5). The paper showed that the major efforts in flow-based detection concentrate on DoS, scan and worm detection, while Botnet detection appears to be a more recent research field.

Figure 13 also identifies some open issues that should be addressed in future research. First, the emphasis will be on improving the *local detection* of threats. An example would be to extend the current research to the detection of unsolicited e-mail. For this, as far as we know, only sporadic flow-based contributions have been proposed [83]. Detection of SPAM sources would address one of the most serious issues in our networks, since it has been estimated that the percentage of SPAM in the first half of 2008 has been 75-85% [84]. Moreover, Ramachandran *et al.* [85] estimated that $\sim$80% of the spam messages are sent by Botnets. In our opinion, Botnet detection is the second major research challenge. Botnet detection will involve long-term analysis of *wide infrastructures* as well as integration of multiple detection methods (Botnets are the source of diverse attacks). Since Botnets are naturally spread over multiple networks, a single monitoring point will probably not be sufficient for detection. To overcome this problem, a third area for future work is the development of *distributed flow-based detection* systems. Distributed detection is particularly important, also because the amount of traffic on high-speed network is still increasing, suggesting that scalability will remain an issue in the future.

4.3 Detecting spam at the network level

Spam is a problem that all Internet users experience in their everyday lives. Symantec Corporation estimates that over 80% of all emails sent in 2008 were spam, a trend that, with a touch of irony, the company considers to be “normal” [84]. The reason we are constantly flooded with unsolicited messages is that spam is profitable. As such, spam detection is likely to remain an “open battlefield” in the coming years.

Nowadays, the most common countermeasures against spam are spam filters. Mail servers usually host the core of spam filtering operations: tools such as Spamassassin [86] reject or accept email messages based on their content. Moreover, many mail clients also locally scan the user’s inbox. However, spam messages are designed to look similar to legitimate emails: examples are “phishing” emails that ask you to provide your bank details. Such camouflaging behavior reduces the effectiveness of content-based methods.

We propose a spam detection approach that does not rely on content information. More specifically, our contribution is based on network flows, defined as “a set of IP packets passing an observation point in the network during a certain time interval and having a set of common properties” [13]. These common properties typically include source/destination addresses/ports and protocol type, and they unequivocally define a flow. Flows have recently received great attention in the research community [87], since they allow scalable network monitoring of large infrastructures. Flows typically only report information about the amount of packets and bytes exchanged during a connection, but nothing about the content of the communication.

In this context, spam detection is a challenge. This section aims to address the following question: *Is it possible to detect hosts from which spam originates by using just flow data?* More specifically, we want to investigate (a) if spam differs from legitimate SMTP traffic at the flow level and (b) how to detect spam at the flow level.

The general assumption in the research community is that a spammer host will behave differently from a legitimate mail server [88, 89, 90]. Capturing this behavior at the network level can lead to the development of powerful tools for early spam detection, easing both the server-side load and the filtering in the client. One contribution in this field is the work of Desikan et al. [91], in which the analysis of time-evolving SMTP connection graphs helps distinguish between mail servers and spammers. A different approach is that taken by Ramachandran et al. [88]. The authors' assumption is that the network behavioral patterns of a spamming host are far less variable than the spam content itself. They therefore propose a spam detection approach based on automatic clustering and classification of sender IP addresses that show a similar behavior over a short observation time. More attention to flow approaches has been given in the works of Schatzmann et al. [89, 90] and Cheng et al. [92]. In [89, 90], the authors suggest that the average number of bytes, packets and bytes/packets of failed, rejected and accepted connections are flow properties suitable for the classification of spam flows. The authors rely on server logs for flow classification. On the other hand, in [92], the authors propose an alternative definition of flows that allows the stateful analysis of spam traffic. Finally, Žádník et al. [93] propose the use of classification trees for spam identification based on flow characteristic.

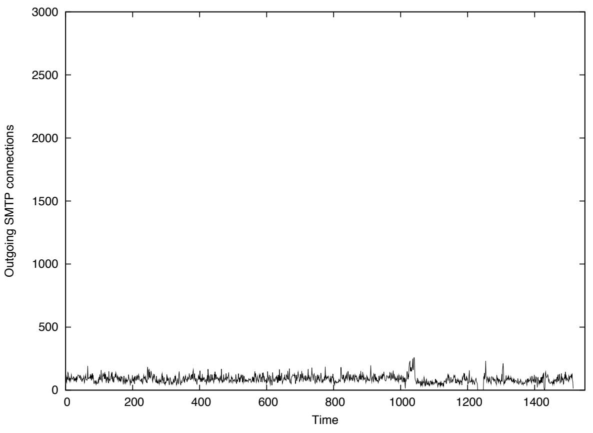
Compared to the previously mentioned contributions, we propose a spam detection algorithm that relies on Netflow compatible flow data and allows the detection of spamming hosts based on just network characteristics.

This section is organized as follows. Section 4.3.1 describes SMTP traffic from a flow perspective, highlighting the differences between a normal and a suspicious host. In Section 4.3.2 we present our spam detection algorithm, followed by a validation of our approach in Section 4.3.3. Conclusions are drawn in Section 4.3.4.

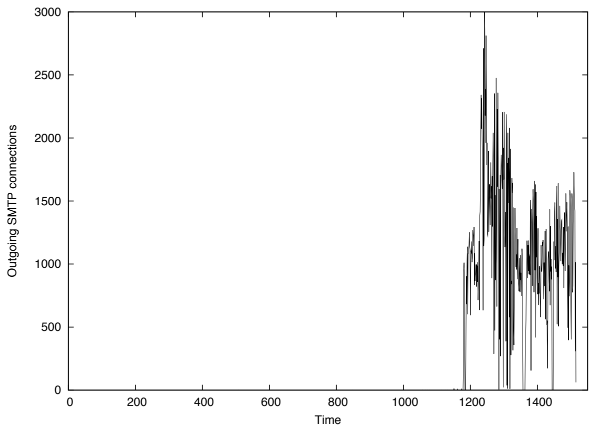
4.3.1 SMTP traffic at the flow level

It is a common assumption that a spamming host's behavior will differ from legitimate SMTP servers. Yet it is interesting to see if this assumption holds in real traffic.

The University of Twente, for example, relies on a system of five load-balanced mail servers, all of them having a similar behavior. Figure 14(a) shows the outgoing SMTP traffic time-series of one of them. Each time slot on the x-axis corresponds to a 5 minutes interval, for a total of five days of observation. There is one main aspect in the mail server behavior. The mail server presents a rather constant activity baseline at around 100 connections per time slot that rarely rises above 250 connections per time slot. This aspect is very significant in our case since it shows that a legitimate mail server is characterized by a steady level of usage. Figure 14(b), on the other hand, shows the outgoing SMTP traffic time series for a host known to have sent spam. Its network behavior is totally different



(a)



(b)

Figure 14: Flow level behavior for a university mail server (a) and a suspicious machine (b)

from the one in Figure 14(a): the time series is characterized by sudden and prolonged activity peaks and a long period in which there is no traffic. Moreover, no usage baseline is present, at the contrary of the mail server. A deeper analysis of the spammer host behavior also reveals that there is no incoming traffic, suggesting that the host has no real traffic exchanges. This behavior is commonly observed in other hosts that have sent spam.

This example suggests that the behavior of suspicious hosts differs substantially from that of legitimate mail servers. Parameters such the incoming and outgoing traffic, as well as the widely variable level of usage can be useful in defining an algorithm for automatic spam detection.

4.3.2 An algorithm for spam detection

In the previous section, we showed qualitatively that the network behavior of suspicious and legitimate hosts could be very different. We now propose an algorithm that will detect, based on just flow information, hosts that are most likely to be spammers. The algorithm consists of two main phases and a post-processing step. In the first phase, hosts that do not satisfy three basic selection criteria are filtered out. This phase aims to reduce the amount of data to be analyzed and to improve the overall performance of the algorithm. The hosts selected in the first phase are then ranked in the second phase by means of five ordering criteria according to their likelihood of being spammers. Finally, ranked hosts are once again filtered according to a post-processing criterion. The algorithm analyzes the SMTP traffic sent and received from the network that is monitored. Of course, this means spam traffic generated by a spammer outside the monitored network and targeting a different network cannot be considered for the analysis. However, the results show that it is not necessary to have a complete overview of all the traffic generated by a spammer to achieve a good detection level.

Selection criteria The selection criteria allow us to concentrate, in the second phase, on a smaller subset of hosts. Therefore, in order to be further analyzed, a host has to satisfy all the selection criteria. The selection criteria aim to filter out at an early stage the majority the not-malicious clients. These criteria are defined as follows:

SC$_1$ **Number of outgoing connections**: We only select hosts that exhibit a certain level of activity:

$$\text{number of outgoing SMTP connections} > \theta_1 \tag{1}$$

SC$_2$ **Connection ratio**: A host is suspicious if it sends far more than it receives. The connection ratio criterion is defined as:

$$\frac{\text{number incoming SMTP connections}}{\text{number of outgoing SMTP connections}} < \theta_2 \tag{2}$$

SC$_3$ **Number of distinct destinations**: Criterion **SC**$_1$ could also flag as suspicious a host that relies on SMTP as logging mechanism (as a printer, for instance). Such a host would probably not receive any message. Nevertheless, such host would usually report to only a limited number of destinations, while a spammer would typically diversify its destinations. A threshold for the minimum number of distinct destinations is used for discriminating these cases:

$$\text{number of distinct destinations} > \theta_3 \tag{3}$$

Ordering criteria Once the suspicious hosts have been selected by applying the selection criteria, we apply the ordering criteria to rank them according to their likelihood of being spammers. While the selection criteria are combined into a binary decision, the ordering criteria yield values from a to e that are later combined into a total score for each host.

OC$_1$ **Number of incoming connections**: This criterion is a refinement of **SC**$_2$. We assume that spammers are not interested in receiving SMTP connections. Therefore, a host that does not have any incoming connection is more likely to be a spammer than one that has incoming SMTP traffic. The score is calculated as follows:

$$a = \begin{cases} 1 & \text{if number of incoming SMTP connections} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

OC$_2$ **Number of distinct destination**: This criterion is a refinement of **SC**$_3$. We assume that a spammer would try to diversify its destinations. Therefore, hosts with a high number of distinct destinations are suspicious. We define the score b as:

$$b = \begin{cases} 1 & \text{if number of distinct destination servers} > \theta_4 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

OC$_3$ **Percentage of idle time**: We assume that hosts with long idle periods are more suspicious than hosts that communicate more regularly over time. We define the score c as:

$$c = \text{percentage of idle time} \tag{6}$$

Algorithm 1 Spam detection procedure

```
1: procedure SpamDetection($Q$ : host set)
2: $S_1 = \emptyset$; $S_2 = \emptyset$;
3: for all $x \in Q$ ordered by decreasing number of outgoing connections do
4:     if $x$ satisfies $\mathbf{SC_1} \wedge \mathbf{SC_2} \wedge \mathbf{SC_3}$ then
5:         $S_1 := S_1 \cup \{x\}$;
6:     end if
7:     if $|S_1| = n$ then
8:         break;
9:     end if
10: end for
11: for all $y \in S_1$ do
12:     Compute $v := \frac{1}{5} \cdot (a + b + c + d + e)$;
13:     if $c > \gamma$ then
14:         $S_2 := S_2 \cup \{y\}$;
15:     end if
16: end for
17: Order elements in $S_2$ by decreasing value of $v$;
18: return top $m$ elements in $S_2$;
```

OC$_4$ Irregularity in activity: Our studies suggest that a suspicious host tends to have a highly irregular transmission pattern. We assume that a host that has a high standard deviation σ of the number of outgoing SMTP flows per 5 minute time slot is more suspicious than one with a low one. We define the score d as:

$$d = \begin{cases} 1 & \text{if } \sigma > \theta_5 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

OC$_5$ Number of peaks: We assume a suspicious host to show sudden traffic peaks. We define peaks as time slots where the number of outgoing connections is higher than $(\mu + k \cdot \sigma)$. μ and σ are respectively the mean and the standard deviation of the number of outgoing connections per 5 minute time slot for the host, and k is a parameter that influences the sensitivity of the measure. Hosts with a high number of peaks are therefore more suspicious. We define the score e as:

$$e = \begin{cases} 1 & \text{if } |\{\text{slots where connection rate} > (\mu + k \cdot \sigma)\}| > \theta_6 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The detection algorithm Algorithm 1 presents the pseudocode for the detection procedure. As explained earlier, the first phase filters hosts according to the three selection criteria (lines 4 through 6). However, in order to keep the algorithm efficient, we only consider the n most active hosts, in terms of outgoing connections, that satisfy the criteria (lines 3 and 7).

In the second phase, the hosts are scored and ordered according to the ordering criteria (lines 11 through 17). For the overall score v, we calculate the average of the single scores a through e. While ranking the hosts, the algorithm also selects a subset of them that, in conjunction with the ranking, are most likely to be spammers. More specifically, only hosts that are not involved in any traffic exchange for the majority of the observation time γ are considered (line 13). This filtering permits the discrimination between hosts that have a fairly constant behavior and hosts that only transmit in bursts, as for example the hosts in Figure 14.

Finally, the algorithm only reports the m top ranked hosts (line 18). The parameter m allows tuning of the output according to the desired security level.

4.3.3 Experimental results and validation

Validation approach Since we based our algorithm on flows, no information about the content of the SMTP connections is available. We therefore need to rely on external services in order to evaluate our results. DNS blacklists (DNSBL) are Internet services that publish lists of offending IP addresses: in our context, IPs that have been involved in spamming activities. Spam DNSBL are repositories which content is likely to rapidly change over time: indeed, a blacklisted host can be rehabilitated if it is no longer involved in spamming activities for a sufficiently long period. Iverson [94] periodically monitors the most commonly used DNSBL and reports on their reliability.

We selected five DNSBL as trusted sources for validation: `zen.spamhaus.org`, `bl.spamcop.net`, `safe.dnsbl.sorbs.net`, `psbl.surriel.com`, and `dnsbl.njabl.org`. We chose this set of DNSBL because they clearly indicate under which conditions a host is going to be added and removed from the list. We define a host to be *positively validated* if it has been blacklisted in at least one of the five DNSBL we are considering.

Experimental settings and results We evaluate our algorithm over three data sets collected at the University of Twente: a reference data set used to developed the algorithm and two newly collected data sets referred as *Set 1* and *Set 2* in the following. Each data set spans over a period of seven days, with an average of ∼15M flows. The time windows over which the data sets span are not overlapping. The implementation of our approach uses SQL scripts and can process a data set in a period of 5 hours.

In our experiments, we measure the *accuracy* of the method, defined as:

$$accuracy = \frac{|\{\text{positively validated}\}|}{m} \tag{9}$$

where m is the number of hosts reported as output by the algorithm and it can be set according to the desired security level. We decided not to compute the false positive and false negative rate since it is not possible to establish a ground truth. For the hosts that we report as suspicious and that are not listed in any DNSBL, indeed, we are unable to say if they are (a) spammers that are not yet listed (true positive) or (b) normal hosts (false positive).

Table 2 shows which parameter values have been used in the experiments. The parameters have been manually tuned based on the statistical properties of the reference data set. We measured that only 5% of the hosts we analyzed have more than 200 connections ($\theta_1 = 200$). In a similar way, only 1% of the hosts have more than 10 distinct destinations ($\theta_4 = 10$). Less than 20% of the hosts present a standard deviation $\sigma > 1$ of the number

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
θ_1	200	θ_2	0.005	θ_3	5	θ_4	10
θ_5	1	θ_6	50	k	5	γ	80%

Table 2: Criteria parameter settings chosen for the experiments

Data set	Time window	Accuracy
Reference set	18 – 24 November 2008	99%
Set 1	2–7 April 2009	96%
Set 2	8–14 April 2009	81%

Table 3: Detection accuracy for the considered data sets.

of outgoing connections per time slot ($\theta_5 = 1$). Moreover, only 1% of the hosts have more that 50 peaks (for $k = 5$, $\theta_6 = 50$). The remaining parameters are specific to the network we are analyzing. In particular, as said in Section 4.3.1, the University of Twente relies on 5 mail servers. Therefore, we set $\theta_3 = 5$. In our network, high volume SMTP sources might receive a limited amount of incoming connections ($\theta_2 = 0.005$) and, for an observation window of seven days, spammers have shown to be idle for at least 80% of the time ($\gamma = 80\%$). Finally, the algorithm selects n = 20,000 hosts that satisfy the selection criteria and outputs the top m = 100 hosts according to the ordering criteria.

Our experimental results show that, on average, the accuracy of the system is 92%. Table 3 presents the detection accuracy for each of the considered data sets. We observe that our algorithm reaches an overall accuracy of 99% in the reference data set, while the accuracy slowly decreases in the newly collected data sets. This phenomenon suggests that the parameters chosen for our experiments might need to be periodically re-tuned according to spam flow characteristics.

Criteria impact In Section 4.3.2 we introduced the criteria we used in our detection algorithm. We now evaluate the impact of each single criterion to the overall detection accuracy of the algorithm. We start evaluating the impact of the only selection criteria **SC**$_1$ and incrementally add one criterion at each run. We measure the overall accuracy on the data set *Set 1* and *Set 2* presented in Table 3. Figure 15 shows the average trend of the accuracy curve with respect to the number of applied criteria. The error bars indicate the standard deviation of the number of validated hosts w.r.t. *Set 1* and *Set 2*. Selection criteria **SC**$_1$ and ordering criteria **OC**$_3$ have the most impact on the accuracy measure, meaning that a high number of connections in a short period of time (bursts) is a key characteristic of a spamming host. Moreover, the accuracy measure presents an increasing trend, meaning that each criterion is beneficial to the detection process. The only exception is **OC**$_5$: in data set *Set 1*, indeed, the criterion forces a decrease of the accuracy, suggesting that under certain condition this criterion may report false positives (legitimate hosts flagged as spammers).

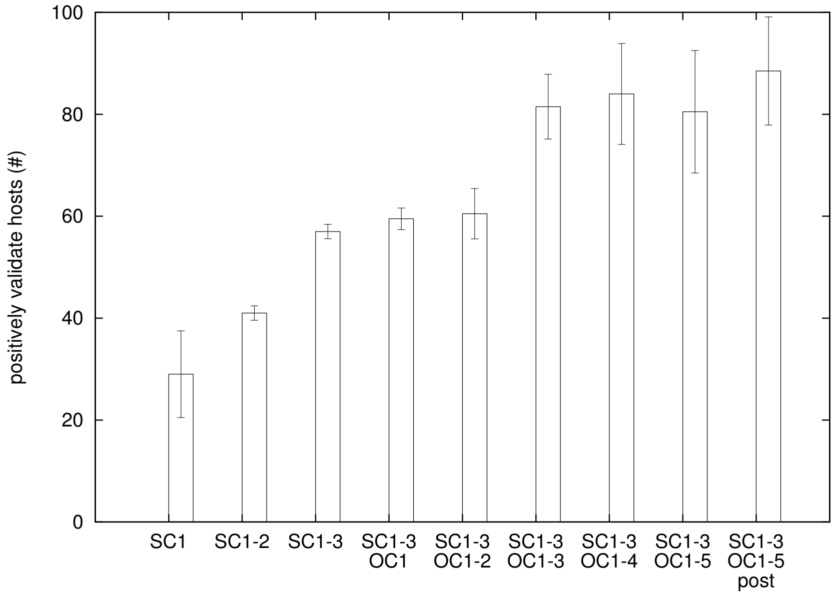


Figure 15: Impact of the selection and ordering criteria on the overall accuracy

4.3.4 Conclusions

This section has investigated if it is possible to detect spammers at the flow level, without relying on email content. Our findings show that the network behavior of suspicious hosts differs substantially from that of a legitimate mail server, both in activity level and incoming/outgoing traffic patterns. Based on these observations, we propose a detection algorithm that makes use of just flow information. Our algorithm has been validated using trusted blacklisting services. The results show that we can detect spamming machines with a 92% accuracy for the traces on which we validate our approach, meaning that the algorithm has a low probability to report false positives (host that are not spammers, but they are flagged as such).

Our work is a first step in flow-based spam detection. In the future, we are interested in assessing the *completeness* of our system, in terms of undetected spamming hosts (false negatives). Moreover, we plan to study how our approach behaves in the presence of very peculiar services, for example a server that is only used for mailing lists. It might happen, indeed, that such systems rank high according to our algorithm, suggesting that other metrics can be added to filter them out. We are also interested in extending our approach to different scenarios, for example Botnet detection.

4.4 Hidden Markov Model modeling of SSH brute force attacks

Since the last decade, we are facing a constant rise of both network load and speed. This has become a problem for packet-based network intrusion detection systems since a deep inspection of the packet payloads is often not feasible in high-speed networks. In this context, flow-based intrusion detection emerged as an alternative to packet-based solutions [95]. Dealing with aggregated network measures, such as flows, helps in reducing the amount of data to be analyzed. A flow is defined as “a set of IP packets passing an observation point in the network during a certain time interval and having a set of common properties" [13]. A flow carries information about the source/destination IP addresses/ports involved in the communication, but nothing is known about the content of the communication itself.

Flow-based time series are a well-known way of visualizing network information [96]. Flow-based time series allow data processing in a streaming fashion, offer a compact representation of network traffic and allow to analyze data keeping into account the temporal relations between events. The drawback of flow-based time series is that we cannot have direct evidence of when an attack happened. In most cases, a ground truth is missing. Attack-labeled flow data sets are rare and their creation is a lengthy and time consuming process [97]. To overcome this problem, approaches based on the superposition of real non-malicious traffic with synthetic attack traffic have been introduced [98, 99]. For these approaches to work, we need models of network attacks.

In this section, we propose a time-series based model of SSH brute force attacks built upon the concept of Hidden Markov Models. We show that our model is able to emulate important aspects of the network behavior of such attacks and generates meaningful flow-based traffic time series.

This section is organized as follows. Section 4.4.1 summarizes the related work on modeling of malicious traffic. Section 4.4.2 describes how SSH brute force attacks are characterized at flow level. Section 4.4.3 presents our model for SSH malicious traffic. The model is evaluated in Section 4.4.4. Finally, conclusions are drawn in Section 4.4.5.

4.4.1 Related work

Hidden Markov Models (HMM) are effective in modeling sequential data [100]. Since they have been introduced in the early 1970s [101], they have been successfully applied to different scientific fields. Examples are biological sequence analysis [102], speech recognition [103] and pattern recognition [104].

Hidden Markov Model triggered the curiosity of many researchers also in the fields of Networking and Intrusion Detection. HMM can be trained on real data and their main characteristic is the ability to capture the temporal behavior of the observed processes. The work of [105] proposes to formalize traffic exchange in terms of "HMM profiles", a stochastic structure suited for sequence alignment. The results show that the models are able to classify traffic sequences at application level. In [106, 107], the authors propose a packet-level model of traffic sources based on HMM. The model proves to be effective in application classification. Moreover, a second fruitful application of the model in [106, 107] is in traffic prediction, namely forecast of short-term future traffic behavior. Similarly to these last contributions, we will use our model for traffic generation. Nevertheless, the work in [105, 106, 107] focuses on the packet-level, while we are interested in flow-based time series.

Hidden Markov Models are particularly appealing in Intrusion Detection, since they are able to calculate how likely a certain sequence of events is. Behavioral models for host-based intrusion detection have been proposed in [108] and [109]. The authors profile the normal sequence of system calls and raise alarms whenever a sequence is unlikely to be seen. Contrary to these contributions, we model malicious activities based on network data. A different approach is the one in [110], where the hidden states model the safety status of a network. Similarly to them, we also assigned semantic meaning to the hidden states. However, in our case the hidden states model the behavior of an attacker.

SSH brute force attacks are a well known cyber-threat [111, 112]. Although the attack is very common, it is still potentially dangerous. Our studies [97] showed that newly set up vulnerable hosts can be compromised within few days and be used as platform for the same attacks. We also showed that SSH attacks are visible at flow level as peaks in the SSH flow time series [66]. However, this observation tells us only how SSH attacks affect the total network traffic when such attacks are at their peak of intensity. In our work, we want to explore how the entire time series generated by a single attacker evolves in time.

4.4.2 Flow-based characterization of SSH brute force attacks

Brute force SSH attacks are one of the most common threats in cyber space [112]. In this section we qualitatively characterize such attacks at flow level, namely describing what a brute force SSH attack looks like if only flow information is available. We analyze the traffic generated by a host known to have performed a SSH brute force attack against our university network. The attack took place in the early afternoon of July 16, 2008 and lasted approximately 40 minutes. During this interval, approximately 8300 distinct university hosts have been attacked. The attack generated a volume of traffic of approximately 32,400 flows, 279,000 packets and 30.5MB.

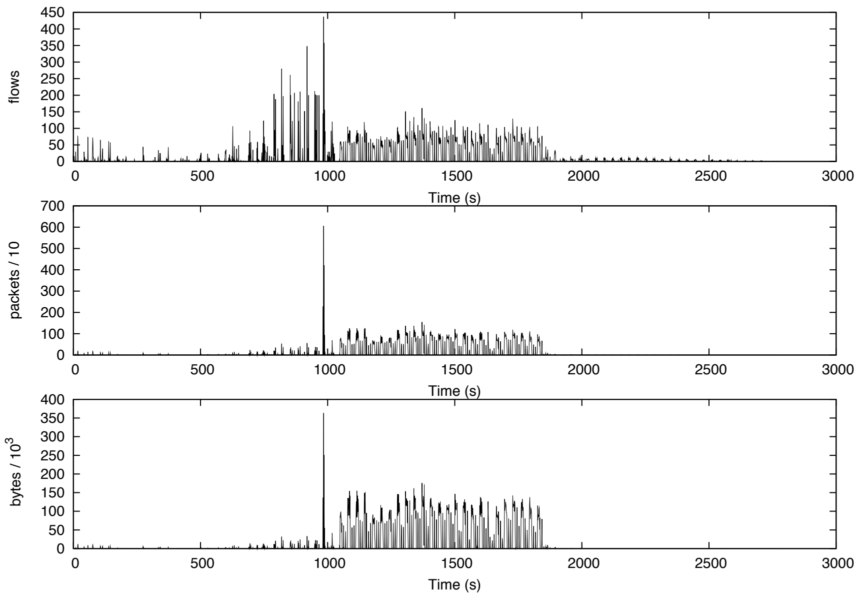


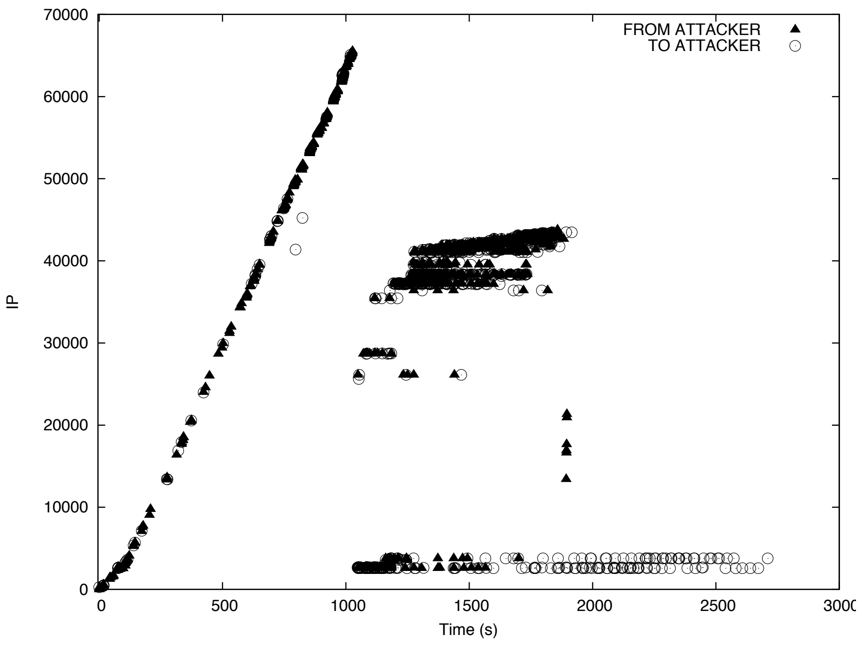
Figure 16: Flow, packet and bytes time series for a malicious SSH user

Figure 16 shows for the attacker the evolution over time of (i) the number of flows created per second, (ii) the number of packets transferred per second, and (iii) the number of bytes transferred per second. The time resolution of the time series is 1 second. Each value in the time series accounts for both the traffic generated by the attacker and the traffic that he receives from the victims. In this way, it is possible to characterize in the same time series the entire attack. During the attack, its intensity varies. In the flow time series we can see that in about the first 1000 seconds the attack intensity grows, reaching a peak of 450 flows/s. After that, the number of flows per second drops abruptly and roughly stabilizes around 100 flows/s. Finally, the attack activity slowly fades off in the last 500 seconds. Moreover, a deeper analysis of the flow time series shows that the activity pattern is not constant in time: each second of activity is often followed by one or more seconds of inactivity. The time series of packets and bytes closely follow the one of flows. Both show a peak around 1000 seconds since the beginning of the attack. As for flows, the trend of packets and bytes tends to stabilize for a while before the attack slowly dies. This behavior suggests that during an SSH brute force scan, the flows, packets and bytes statistics are mutually correlated.

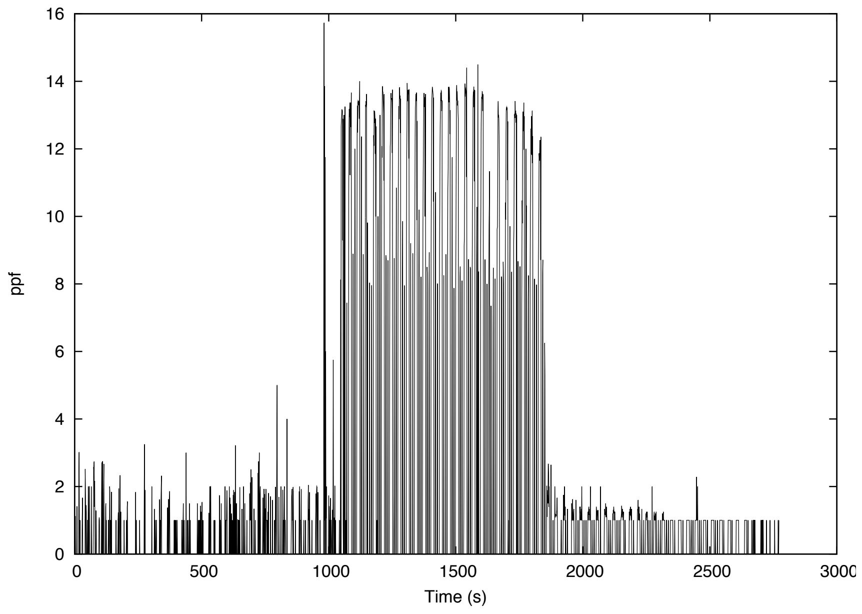
A different view on the attacks is given by Figure 17(a). Each mark in the graph either represents a malicious connection from the attacker to a victim or the answering connection from the victim back to the attacker. The y-axis gives the 65,535 possible destination addresses in the university network. We identify *three attack phases*: During the *scanning phase* (first 1000 seconds), the attacker performs a sequential SSH scan spanning over the entire network address space. In this phases, the attacker gathers information on which hosts run a vulnerable SSH service. Only few victims respond to the attack. Once this phase is completed, the attacker initiates a brute force user/password guessing attack (*brute force phase*). In this phase, only a small subset of the hosts in the network is involved. This phase corresponds to the second block of 1000 seconds and is characterized by a high interaction between the attacker and the victims. Finally, after about 2000 seconds since the beginning of the attack, the brute force phase ends. Nevertheless, the time series in the previous picture have already shown that after this moment in time there is still traffic. Form Figure 17(a) it is now evident that the residual traffic is due to compromised hosts that communicate with the attacker. We refer to this final phase as the *die-off phase*.

Although the three attack phases are clearly visible in Figure 17(a), they are not so clearly identifiable from the flow, packets and bytes time series shown in Figure 16. However, the fact that the three time series are correlated allows us to derive a more suitable measure. Figure 17(b) shows the evolution over time of the *packets per flow*, again with a resolution of 1 second. Using this measure, the three phases are clearly visible. The scanning phase is characterized by only few packets per flow, in average between 1 and 1.5. These values are consistent with a scenario in which several three-way handshakes are initiated but only few are completed. When the brute force phase starts, the number of packets per flow has a sharp rise: from 1.5 to a average of about 11. During this phase, several user/password combinations are tested against the same victim. This explains why the attacker produces a higher number of packets per flow. Finally, the die-off phase sees again only few packets per flow. In the majority of the cases, we observe only one packet per flow. The variation of packets per flow over time seems therefore to be a key characteristic of the behavior of an SSH brute force attacker. It moreover shows that the flow, packets and byte time series still carry enough information to characterize the attack.

The attack behavior that we described in this section is typical for a brute force SSH attack. While monitoring the university network, we observe in average one attack with these characteristics per day. In the following section, we will describe how the flow-based



(a)



(b)

Figure 17: Temporal visualization of a brute force SSH scan (a) and variation of packets per flow during a scan (b)

characterization that we just proposed can be used to model the behavior of an SSH brute force attacker.

4.4.3 Modeling with HMM

The analysis of a typical SSH brute force attack that we presented in Section 4.4.2 pointed out three key characteristics of such attacks:

1. The flow, packet and byte time series exhibit a clear correlation.
2. Attacks consist of three phases: a scanning phase, a brute-force phase and a die-off phase.
3. The subdivision into phases may not be evident when we observe the flow, packet and byte time series directly, but it becomes manifest when we consider the packet per flow time series.

These key characteristics will play a central role in the following when modeling attacks using Hidden Markov Models (HMM).

This section is organized as follows. Section 4.4.3 briefly recapitulates the definition of HMM. In Section 4.4.3 we show how an SSH brute force attack can be described as a Markov Chain. In Section 4.4.3 we describe the output probabilities associated to each state and how they can be used to generate meaningful time series. Finally, Section 4.4.3 explains how the model parameters are computed from real data traces.

Hidden Markov Models

Hidden Markov Models are a class of statistical models able to describe sequences of data resulting from the interaction of several random processes.

Formally, an HMM is a discrete time markov chain (DTMC) where each state is augmented with a probability distribution over a finite set of output symbols. Given a sequence of states $Q = q_1 q_2 \dots$ with associated output symbols $K = k_1 k_2 \dots$ we say Q forms the *hidden sequence* and K forms the *observation sequence*.

With $S = \{s_1, \dots, s_n\}$ we denote the finite set of hidden states. S is called *hidden chain*. With q_t we denote the state at time t. With a_{ij} we denote the probability of jumping from state s_i to state s_j. Since we are dealing with a DTMC, this probability only depends on the current state s_i, i.e.: $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$. With π_i we denote the probability of the initial state being s_i, i.e.: $\pi_i = P(q_1 = s_i)$. With $O = \{o_1, \dots o_m\}$ we denote the finite set of output symbols. With k_t we denote the output symbol seen at time t. With $b_i(o)$ we denote the probability of seeing output symbol o when the hidden state is s_i, i.e.: $b_i(o) = P(k_t = o \mid q_t = s_i)$.

An HMM separates the state chain from the observable output. This key characteristic allows us to model malicious SSH attacks in an effective way and to generate synthetic flow, packet and byte time series.

The hidden chain

In Section 4.4.2, we explained that an SSH brute force attack consists of three phases: a scanning phase, a brute-force phase and a die-off phase. We make use of these phases to define the hidden chain.

Our model consists of the following seven states:

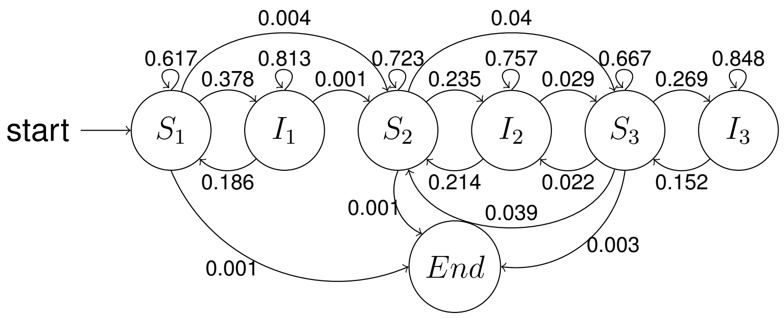


Figure 18: HMM for the SSH brute force attack with an example of transition probabilities learnt from real data

- the states S_i, $i = 1, 2, 3$. In these states, the attacker is *active* and causes network traffic.
- the states I_i, $i = 1, 2, 3$. In these states, the attacker is *temporary inactive*, as described in Section 4.4.2.
- the end state End.

The state S_1 is the start state with $\pi_{S_1} = 1$. Figure 18 depicts the states and the possible transitions.

The states S_1 and I_1 model the scanning phase of the attack. As it can be seen in Figure 18, once the attack moves from the scanning phase to the brute force phase, represented by the states S_2 and I_2, it cannot return to the previous phase. This ensures that the scan will not be performed more that once for each attack. On the other hand, the die-off phase (states S_3 and I_3) can partially overlap with the brute force phase. This phenomenon is modeled by making the states $\{S_2, I_2, S_3\}$ a fully connected chain. However, the transition probabilities for this subset of states will privilege transitions in the same phase. Finally, the state End models the end of an attack. We allow the model to jump from each active state S_i to the End state, thus reflecting the fact that some attacks stop after the scan phase or the brute-force phase.

The output probabilities

The aim of our model is to generate meaningful synthetic flow, packets and bytes time series for a SSH brute force attack. Hence, at each transition, our model should output a triple (F, P, B) with the values for the three time series.

It is important to note that these three values are not independent, as shown in Section 4.4.2. Hence, to generate correctly correlated values for the three time series, a joint output probability distribution $P_{F,P,B}$ would be needed for each state of the model. In the following, we will present a different approach that approximates the triple-joint probability distribution $P_{F,P,B}$.

To each active state S_i, $i = 1, 2, 3$ in our model we assign the following two distributions:

1. an empirical probability distribution of flows P_F;
2. an empirical joint probability distribution of packets per flow (PPF) and bytes per packet (BPP), denoted as $P_{PPF,BPP}$.

At each transition, random values of F, PPF and BPP are generated according to the empirical distributions associated to the current state. Given the number of flows F, we assume the number of packets per flows and the number of bytes per packets to be the same for all the flows for this emission. We calculate

$$\begin{aligned} P &= PPF \cdot F, \\ B &= BPP \cdot PPF \cdot P. \end{aligned}$$

The joint probability distribution $P_{PPF,BPP}$ and the indirect computation of P and B by the above expression ensure the strong correlation between F, P and B that we have observed in the data.

In the states I_i, $i = 1, 2, 3$, the attacker is by definition temporarily inactive and the triple $(0, 0, 0)$ is the only allowed output.

The parameter estimation

Once the hidden chain and the outputs of the model have been defined, we need to estimate the transition probabilities and the emission probability distributions for the states S_i, $i = 1, 2, 3$. Several methods for estimating the parameters of an HMM have been proposed in literature, for example the Baum-Welch algorithm [101], or the simulated annealing method of [113]. However, these methods are used when the training is based on sequences of observations only and the hidden state sequence is unknown.

In our training procedure, we follow a different approach. The analysis of the packets per flow time series, such as the one in Figure 17(b), offers us a way to precisely relate each observation in a trace with the hidden state that emitted it. We therefore manually labelled the traces in our training data sets. Once the hidden state sequence is known, we calculate each transition probability as

$$a_{ij} = \frac{|\{\text{transitions from } s_i \text{ to } s_j\}|}{|\{\text{transitions from } s_i\}|}.$$

Figure 18 gives an example of transition probabilities learnt from real data. The hidden state sequence is used to compute the output probabilities associated to each state. We calculate the distributions P_F and $P_{PPF,BPP}$ for a state S_i, $i = 1, 2, 3$ from the frequency histograms of the observations emitted from that state.

4.4.4 Validation

In this section we will evaluate the performance of the model proposed in Section 4.4.3. In particular, we will show that the model is able to generate synthetic traffic that has the same statistical characteristics of the SSH brute force attack traces we used as training.

We based our validation on two data sets consisting of malicious SSH traces collected at the University of Twente network (*original data sets*). The validation proceeds as follows. First, we train an HMM for each distinct data set. Second, we use the models to generate groups of synthetic traces sufficiently large for the calculation of the confidence intervals. We refer to these traces as *synthetic data sets*. Third, we analyze the statistical properties of the synthetic data sets and we compare them with the original data sets. The aim of this

Data Set	Collection time	Traces	Avg. Flows/sec	Avg. Packets/sec	Avg. Bytes/sec
Set 1	13-20 July 2008	17	11.06	66.91	7337.33
Set 2	19-26 April 2009	13	15.80	150.52	19016.00

Table 4: Statistical characteristic of the collected data sets

analysis is to show that the model is able to encode sufficient information to afterwards correctly emulate the original traces.

Original data sets

Our model of SSH brute force attacks has been tested on two data sets. Each data set contains flow, packet and byte time series for a group of hosts known to have scanned our networks. The malicious hosts are all distinct.

The time series have been created considering time slots of 1 second. The volume of traffic for each time slot is comprehensive of both the traffic generated by the scanner and the traffic that it receives.

Table 4 presents the data sets. Both data sets have been collected during a monitoring window of one week on the network of the University of Twente. The offending hosts have been identified by a high interaction honeypot that is normally active in our network. *Set 1* has been collected in July 2008 and consists of 17 traces. *Set 2* has been collected in April 2009 and consists of 13 traces. Other hosts performing SSH malicious activities have not been considered part of the data sets since they appear to belong to a different class of scans. The statistical analysis of the two data sets shows that the average values of flows, packets and bytes over time have changed in time. In *Set 2*, the attackers appear to produce in average more that twice the amount of packets and bytes compared to *Set 1*. This suggests that, while the attack mechanism stays the same, the attacks' intensities are likely to vary in the course of time. As a consequence, models trained on real data would need periodical retrain.

Synthetic trace generation

We define a synthetic trace as the sequence of observations that the model outputs when a random path is taken. The generation process can be summarized as follows. Let's assume the model to be in state s_i:

1. at time t, the model jumps from the current state s_i to the next state s_j according to the transition probabilities a_{ij}, $j = 1, \dots n$.
2. if s_j is the End state, the path is concluded and the trace ends.
3. once s_j has been selected, the model randomly selects F, PPF and BPP.
4. the model outputs the triple (F, P, B), calculated on the basis of the random values generated in the previous step (as explained in Section 4.4.3).
5. once the observations have been emitted, the process iterates from step 1.

At each iteration, the model chooses which triple (F, P, B) will be emitted. This choice is independent from the previous outputs and is controlled only by the empirical probability distributions associated with the current state. The model controls also the duration of a trace, since a trace ends only when a transition to the End state is randomly selected.

Testing methodology

Our testing methodology aims to measure the average statistical characteristics of a set of synthetic traces and compare them to the ones of the original data sets *Set 1* and *Set 2*. Each statistical metric is calculated for flows, packets and bytes. We are interested in three type of statistical measures:

- the mean and standard deviation for flow (μ_F, σ_F), packets (μ_P, σ_P) and bytes (μ_B, σ_B). These measures describe the *overall behavior* of flow, packets and byes independently of each other in a trace.
- the correlation coefficients ρ_{FP}, ρ_{FB} and ρ_{PB} between flows, packets and byes. These measures describe the *dependence* between flows, packets and bytes in the same trace.
- autocorrelation of lag 1 of flows (R_F), packets (R_P) and bytes(R_B). The autocorrelation captures the *evolution* of a trace over time, measuring the interrelation of the trace with itself in different moments of in time.

The previously introduced measures are relative to a single trace. In our experimental results, we calculate the average values of each measure for both the original data sets and the synthetic ones. For the synthetic trace, we also calculate the 95% confidence interval. Each synthetic data set consist of 300 traces. Finally, we evaluate how well the synthetic traces approximate the original ones. In order to do so, we calculate for each measure m the relative error between the original traces and the synthetic ones:

$$Err = \frac{|m_{orig} - m_{syn}|}{m_{orig}}$$

Experimental results

This subsection presents the numerical results obtained from the analysis of the synthetic data sets. Table 5 offers an overview of the average statistical measures for both the original and the synthetic data sets. The same table also lists the relative error between original and synthetic measures. The results will be discussed in the following sections.

Average mean and standard deviation Both the model trained on *Set 1* and the one trained on *Set 2* approximate the averages of flows, packets and bytes within a 10% relative error.

The results also show that our approach approximates the standard deviation of both the original data sets within 10% relative error, with only few exceptions: the average standard deviation of packets for *Set 1* and the average standard deviations of flows and bytes for *Set 2*. Regarding *Set 1*, the synthetic measure underestimates the one in the original data set. On the contrary, in *Set 2*, the synthetic measures are higher than the original. We suspect this phenomenon is related to the autocorrelation of the traces in the original data sets.

	Set 1	Synthetic 1	Err	Set 2	Synthetic 2	Err
μ_F	11.06	12.27 $\pm$ 0.33	0.109	15.80	15.15 $\pm$ 0.65	0.041
μ_P	66.91	66.66 $\pm$ 3.67	0.046	150.52	138.85 $\pm$ 8.5	0.077
μ_B	7337.33	7524.73 $\pm$ 523.11	0.025	19016.00	18107.88 $\pm$ 1153.53	0.04
σ_F	36.45	38.33 $\pm$ 1.12	0.051	40.0	47.01 $\pm$ 1.87	0.17
σ_P	324.29	243.43 $\pm$ 10.91	0.249	379.38	419.16 $\pm$ 16.55	0.10
σ_B	28510.35	28345.60 $\pm$ 1616.63	0.005	47060.07	55378.58 $\pm$ 2239.91	0.17
ρ_{FP}	0.79	0.79 $\pm$ 0.012	0.001	0.83	0.86$\pm$ 0.01	0.039
ρ_{FB}	0.76	0.74 $\pm$ 0.016	0.023	0.79	0.81$\pm$ 0.01	0.024
ρ_{PB}	0.94	0.98$\pm$ 0.002	0.047	0.98	0.98$\pm$ 0.001	0.001
μ_{R_F}	0.46	0.23 $\pm$ 0.009	0.49	0.64	0.26 $\pm$0.01	0.59
μ_{R_P}	0.56	0.25 $\pm$ 0.012	0.54	0.71	0.30 $\pm$ 0.009	0.57
μ_{R_B}	0.58	0.26 $\pm$ 0.012	0.54	0.75	0.30 $\pm$ 0.009	0.59

Table 5: Numerical comparison between the original and the synthetic data sets.

Table 5 also presents the 95% confidence intervals for the average means and the standard deviations. For all measures, the confidence intervals are close to the average values.

Average correlation The correlation coefficients show that the proposed model is able to capture the interrelations between flows, packets and bytes, despite that the realizations of the random variables $\mathbf{F}$ and $\mathbf{ppf}$ are independently drawn. The relative error in the case of the correlation coefficients is indeed less or equal to 4.7% (ρ_{PB}) in the case of *Set 1* and less or equal to 3% (ρ_{FP}) in the case of *Set 2*.

In the same table we listed also the 95% confidence interval for the average correlation coefficients. As in the case of the average relative error, described in the previous section, the confidence intervals are closed to the mean values.

Average autocorrelation The last measure we consider is the average autocorrelation. The autocorrelation characterizes the temporal evolution of a trace.

For both *Set 1* and *Set 2* our model fails to approximate the autocorrelation values. The autocorrelation of the synthetic traces, indeed, is roughly half of the autocorrelation in the original data sets. This means that consecutive values in a synthetic trace have a higher random component than in the original traces.

We believe that the cause of lower autocorrelation coefficients can be found in the attacker behavior during the brute force phase. The original traces, indeed, show that during this phase the time series presents a certain regularity, as for example a bounded number of flows per seconds. Our model, on the other hand, randomly selects at each iteration new values for flows, packets and bytes, without any memory of the previous outputs. This behavior is reflected in lower autocorrelation values. We consider to extend the model to capture regularity in the brute force phase as a possible future work.

As for the previous measures, also in this case the confidence intervals show that the model has a low variability in the autocorrelation values.

4.4.5 Conclusions

We have presented a compact model of SSH brute force attacks based on Hidden Markov Models. The model has been inferred on the basis of only flow information and it encodes the network behavior of SSH attacks: scanning phase, brute force phase and die-off phase. The model parameters have been calculated on the basis of real data traces captured at the University of Twente network.

We also demonstrate that the model, once trained on real data, is able to emulate the network behavior of a SSH brute force attacker. Synthetic traces approximate the mean, standard deviation and correlation of flow, packets and byte time series within 10% relative error. The model fail only in approximating the autocorrelation. The synthetic traces, indeed, seems to have a higher random component than the original training trace.

As far as we are aware, this was the first time that HMM have been applied to the generation of flow-based time series for malicious users. The results are encouraging, but many aspects are open for future work. First, we aim to refine the model. For example, a more detailed model of the brute force phase can improve the autocorrelation. In addition, the empirical emission distributions can be substituted by estimated distribution functions to make the model resilient to unforeseen observations. Second, we plan to adapt the model to be used for detection. In this context, we are also interested in investigating if the model we proposed is suitable for detection of other brute force attacks that show a similar phase behavior. An example can be a brute force attack against the telnet service. Third, we want to apply our HMM approach to other attack types, such as DoS attacks or worms.

4.5 Traffic characterization

The goal of this PSNC activity was to analyze international network traffic of a research network, the PIONIER Polish Optical Internet, based on collected Netflow data and on data obtained by optical passive monitoring.

Section 4.5.1 describes the measurements environment and the collected data. Section 4.5.2 analyzes NetFlow traffic with regard to ports and applications and then correlates the results with those obtained from optical passive monitoring. Section 4.5.3 compares the two methods of data analysis for a short-time scale network monitoring. Finally in Section 4.5.4 we look at traffic periodicity on the international link of PIONIER.

4.5.1 Collection methodology

NetFlow

We used NetFlow data coming from the PIONIER NREN border router where the European research network GEANT link is terminated. This was NetFlow v5 sampled 1:10000.

The NetFlow data presented here was collected for a period of three weeks, from 11 June 2009 to 02 July 2009. During this period we processed about 7.3 GB of data.

Optical passive monitoring

We used an optical passive monitoring sensor [114] that monitors the research traffic of the Polish NREN on its connection towards European research network GEANT. This equipment monitors all incoming and outgoing traffic on a 10 Gb/s Packet over Sonet link. The deployed infrastructure consists of monitoring stations equipped with monitoring DAG cards for 10 Gb/s links from Endace company. Network traffic monitoring works as follows. Packets are copied from the network (one direction) by an optical splitter. Packets are then captured by an aforementioned network monitoring card. When reached the server packets are processed by DiMAPI (Distributed Monitoring Application Programmable Interface) middleware [115], which is divided into two parts: the `mapid` and `mapicommd` daemons running on monitoring stations with monitoring cards and DiMAPI libraries, which are linked to applications, which can run on the same or different PCs than the monitoring stations, typically on application PCs in order to use their separate processing power.

The passive monitoring system we used runs `ABW` - an application to generate traffic classification based on packets inspected by DiMAPI middleware [116]. ABW is a Web tool run from any Web browser which uses graphical interface to show used capacity in a wide range of time resolutions (from one second to months), show distribution of used capacity into protocols and applications including those that use dynamic ports. It can monitor how the real traffic on a network link is composed from protocols at different layers of the OSI hierarchy or what user-specified subsets of traffic based on arbitrary L2 to L4 header fields are present on the link. It identifies many commonly used protocols that use dynamically allocated ports, such as passive FTP, HTTP (which can use other ports than 80 and 443), Skype telephony or file-sharing protocols, such as BitTorrent, Gnutella or DC++. It also monitors the volume of multicast and IPv6 traffic as two interesting subsets of traffic. The volume of traffic (separately the number of packets and the number of bytes) for all configured protocols is measured every second and stored into an RRD database where the results are gradually aggregated. Upon user request, the application generates graphs for specified monitored links, protocols, time intervals and time resolutions (predefined or user-defined). Currently the following resolutions are available: last 10 mins (interval 1 sec, avg/max 30 sec), last 1 hour (interval 10 sec, avg/max 3 min), last 10 hours (interval 1 min, avg/max 20 min), last 1 day (interval 3 min, avg/max 1 hour), last 1 week (interval 20 min, avg/max 6 hour), last 1 month (interval 3 hour, avg/max 1 day), last 1 year (interval 1 day, avg/max 1 month). The application is split into two phases: the continuous data acquisition and the data presentation. The graphs also show the average and maximum link load for the same time period, but with a coarser time resolution.

The data presented here were collected by the passive monitoring system for a period of three weeks, from 11 June 2009 to 02 July 2009.

4.5.2 NetFlow traffic characteristics

Protocols and ports Within the period of 3 weeks of data collection passing through our router we were able to point out 10 most popular ports network traffic was related to. Table 6 lists those ports identified in the whole traffic observed in the PIONIER border router. Those ports may indicate most popular applications running on client and server machines within analyzed networks. This should not, however, be treated as an exact study of network traffic payload, as this would require much deeper packet inspection

Port Number	Typical Role
22	SSH - The Secure Shell Protocol
25	SMTP - Simple Mail Transfer Protocol
53	DNS - Domain Name Server
80	HTTP - Hypertext Tranfer Protocol
119	NNTP - Network News Transfer Protocol
443	HTTP over SSL/TLS
554	RTSP - Real Time Streaming Protocol
1094	rootd protocol
1935	Macromedia Flash Communications Server MX
4662	eDonkey P2P Network
6881	Typically used by BitTorrent applications
8000	Officially iRDMI, most often however used by Shoutcast streaming media
27005	Officially FlexLM, most probably however used by online games

Table 6: Ports observed in the traffic between PIONIER and GEANT

Protocol Number	Protocol Name
1	ICMP - Internec Control Message Protocol
4	IP in IP Encapsulation
6	TCP - Transmission Control Protocol
17	UDP - User Datagram Protocol
41	IPv6 - Internet Protocol version 6
47	GRE - Generic Routing Encapsulation
50	ESP - Encapsulating Security Payload, IPSec related
93	AX.25 protocol

Table 7: Protocols observed in the traffic between PIONIER and GEANT

(layer 7 analysis) which is not possible due to legal conditions. However, knowing the most popular applications available on the consumer market we can correlate these with data we collected to try to identify most popular network applications.

Ports within 1-1024 range are well defined and rarely used for anything else than what the IANA originally assigned them for. Ports outside this range are often used for very different purposes, therefore we can only assume the most likely role of the port in our traces basing the guess on common knowledge about widespread applications and ports they use for online communication.

With regards to protocols, Table 7 presents those which we were able to identify within our Netflow traces.

Results gathered from the edge router and shown in Table 6 point to the most commonly used applications throughout the networks connected to the Internet by PSNC. Although one must take into consideration a relatively high level of results uncertainty due to highly arbitrary port number usage in today’s Internet. While much more sophisticated analysis is required to obtain more detailed results, analyzing NetFlow statistic still provides valuable information about the nature of traffic within a network.

As one can see in Table 6, HTTP traffic to and from port 80 is dominant throughout the network. It is at the same time probably the most ambiguous result, as many different applications tend to hide behind this port for various reasons. Apart from typical webserver communication, often other services use this port to bypass firewall restrictions in different organizations. This may include services such as streaming media (with streaming video becoming more common than ever before, this is particularly probable) of various sorts, VoIP connections or even p2p file sharing networks. Having so many different applications share the same port does not make network management easier, as traffic shaping and/or filtering needs to perform deep packet inspection (such as layer-7 filtering) in order to differentiate between various traffic types.

The second most popular port is most probably related to Shoutcast/Icecast streaming, which - despite the IANA port assignment - most often occupies ports in range between 8000 and 9000, the former being currently a de-facto standard port for best-effort http-based streaming. Much of this media streaming activity, however, is also included in statistics for port 80 (as mentioned before) as well as for port 443 (also ranked high in the top-ten) in order to bypass some restrictive firewalls and/or traffic shaping mechanisms in use in various organizations and/or operators.

Streaming media is also the reason for the port number 554 being the third most popular, as it is most commonly used by RTSP-based streaming media services, most notably RealMedia. As with all other ports, we must note that other traffic might be disguised as RTSP traffic for the same reasons as with port 80.

Overall, streaming media is currently dominating the bandwidth usage, surpassing even typical, non-disguised p2p traffic – but not file-sharing in general, as most probably a significant part of those activities is done using non-dedicated ports and/or other services, such as file hosting servers. Typical p2p-related ports are found relatively rarely throughout the traces, which is a bit surprising, but the explanation for this phenomenon is most probably related to bypassing firewalls, as previously stated.

Applications usage

Then we also looked particularly at the traffic on the link towards GEANT network to observe whether sampled Netflow traces reflect the nature of a traffic passing through the network. One can compare the Netflow analysis results to the very accurate optical passive monitoring which is deployed on this link. Top ten ports (sorted by traffic volume shares) noticed within our Netflow traces are presented in Table 8 and Table 9. Fig. 20 and Fig. 20 show the utilization graphs from optical passive monitoring providing information about applications usage in the traffic. The red part of the graphs indicates error in the analysis application where the basic functionality was not working and classified all traffic to type `Other`. The period of analysis is three weeks of data.

Highest amount of traffic is, surprisingly, directed to port 1094, which is officially considered a rootd port. It is, however, doubtful if this service is in fact responsible for this amount of data. We can only suspect that a file-sharing application or an Internet-worm might be the cause of such traffic volume. This, however, cannot be verified without performing a layer 7 analysis, which is beyond the scope of this document. This traffic, however, is most probably a dominating component of `Other` in the optical passive monitoring graphs.

Optical passive monitoring graphs do indicate that a high level of HTTP and FTP traffic (either inbound or outbound) contributes significantly to the overall bandwidth consumption.

port	flows	octets	packets
1094	8.825	11.015	7.904
80	5.899	6.774	5.491
2128	3.477	2.836	2.359
20000	0.894	2.374	2.138
20001	0.655	2.294	1.837
22	1.402	1.358	1.596
873	0.311	0.999	0.683
20002	0.375	0.984	0.865
20003	0.275	0.683	0.596
20	0.270	0.455	0.306

Table 8: Observations of shares of ports usage from GEANT to PIONIER

port	flows	octets	packets
1094	6.341	4.148	6.279
80	5.434	3.612	4.832
20000	1.118	2.663	1.817
20001	0.688	1.578	1.293
22	1.484	1.574	1.833
20002	0.406	0.977	0.707
2128	2.344	0.933	1.634
2170	0.899	0.853	0.624
20003	0.297	0.626	0.464
20004	0.246	0.562	0.382
8000	0.508	0.534	0.356

Table 9: Observations of shares of ports usage from PIONIER to GEANT

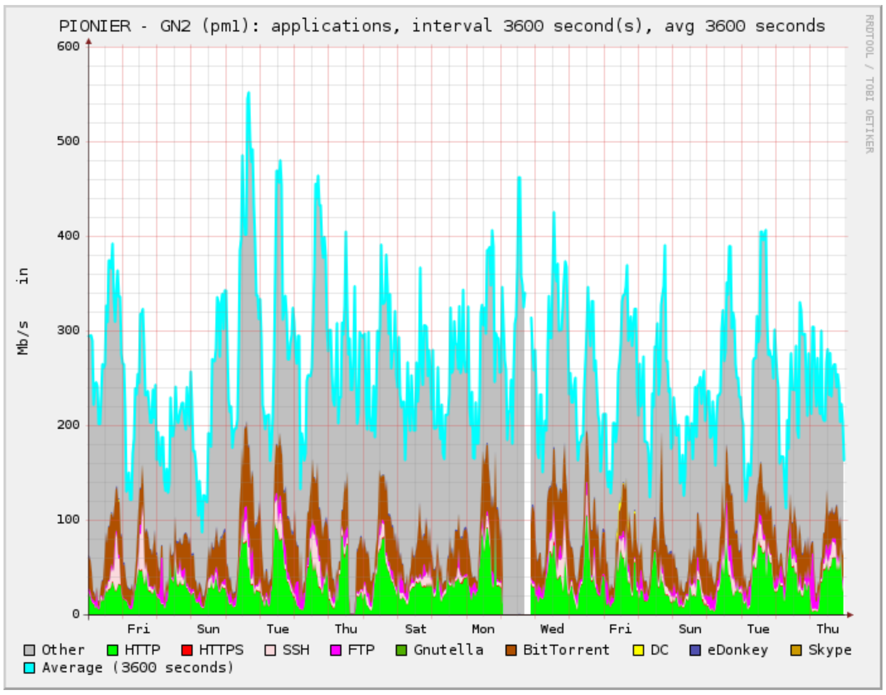


Figure 19: Share of incoming traffic observed with optical passive monitoring

This is consistent with Netflow-based analysis results - in this case also relevant ports are found on the top-10 lists. This is also the case with SSH port 22 - especially in the PIONIER to GEANT direction (noticeable both on relevant top-10 list and graph). Most of this load is probably due to SCP operations, as a typical SSH session does not generate high traffic volume. Other than that, Shoutcast/Icecast streaming is worth mentioning as a significant contributor to overall bandwidth consumption, again – mostly related to PIONIER to GEANT direction. Other ports are most probably responsible for BitTorrent (and other p2p) traffic, which is again consistent with results obtained from passive monitoring.

Comparing protocol graphs obtained from passive monitoring with Netflow analysis results one can only tell that TCP is the dominant type of traffic registered, with UDP contributing to a much lesser extent. There is also a relatively constant IPv6 component. ICMP is the last noticeable protocol in this scope. All these results are consistent between Netflow and optical passive monitoring results.

4.5.3 Sampled NetFlow vs optical passive monitoring experience

Network infrastructure monitoring usually uses information obtained from network components, such as routers and switches. One of the the most common protocols here is Netflow used to provide useful statistics about network traffic. However, common limitations are that it provides only statistics about network traffic in coarse time granularity and is highly impacted by sampling. Of course it is possible to run 1:1 NetFlow but it can significantly impact network device performance. The main benefit of the optical passive approach is the accurate measurement of the actual packets faced by user traffic, due to ability of providing un-sampled packet-level traces and packet inspection. It enables identification of the individual traffic flows and high frequency of measurements which can

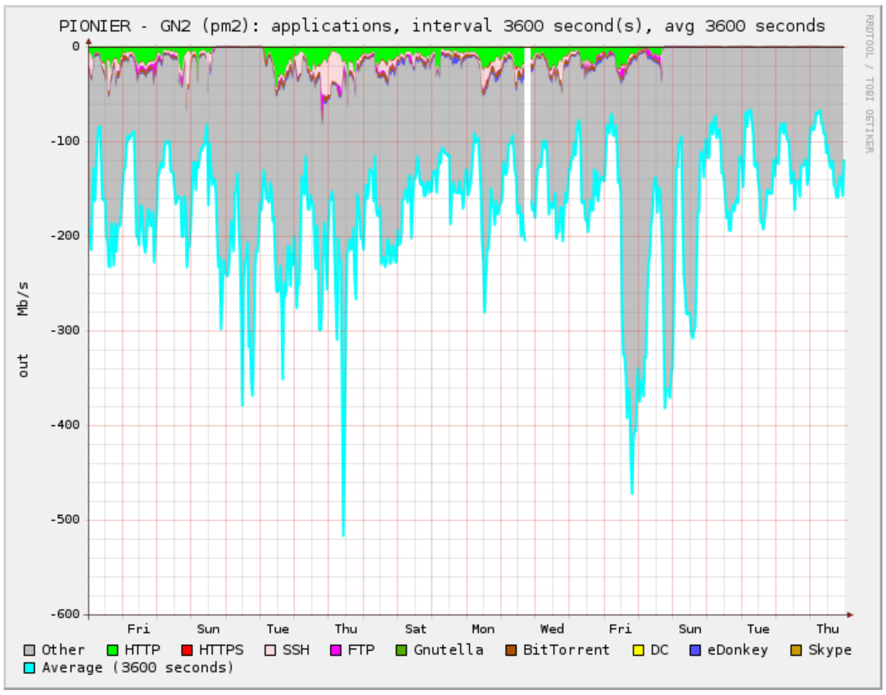


Figure 20: Share of outgoing traffic observed with optical passive monitoring

	TCP		UDP		ICMP		IPv6	
	Packets	Bytes	Packets	Bytes	Packets	Bytes	Packets	Bytes
NetFlow	90,00%	95,11%	9,18%	4,29%	0,151%	0,027%	0,60%	0,56%
Optical passive	93,17%	97,94%	5,59%	1,70%	0,171%	0,018%	3,50%	4,29%
Difference	-3%	-3%	64%	152%	-12%	49%	-83%	-87%

Table 10: Share of incoming traffic observed with NetFlow and optical passive monitoring

better identify short-term network anomalies.

In this section we tried to understand the differences in results collected in PIONIER network by both techniques and the impact of NetFlow sampling on the traffic seen and traffic estimations usefulness. Table 10 and Table 11 show the relationship between share of traffic in the total volume for different protocols measured with the use of NetFlow and optical passive monitoring. They show the data measured on 17 June 2009 within 1 hour with resolution of 10 seconds (for optical passive) and 1/10000 sampling (NetFlow). Such a sampling rate is a common safe setting in Juniper T-series which are not equipped with a dedicated Service PIC enabling increasing the rate of NetFlow sampling without impacting the control plane of the router.

According to both figures the least difference in traffic type is for TCP – about 3% both for number of packets and bytes, which is a very good estimation useful for most of the use cases. Such result corresponds closely to the high volume of this type of traffic seen by both techniques, which given the sampling nature of NetFlow, is expected. The remaining

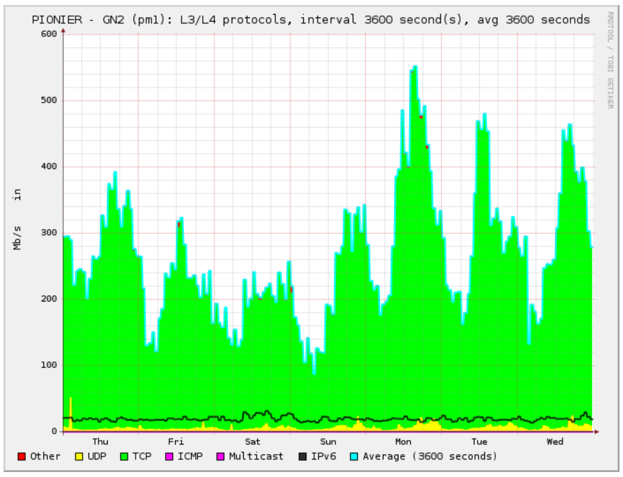
	TCP		UDP		ICMP		IPv6	
	Packets	Bytes	Packets	Bytes	Packets	Bytes	Packets	Bytes
NetFlow	88,26%	92,19%	11,07%	7,21%	0,135%	0,031%	0,377%	0,394%
Optical passive	92,25%	94,98%	6,12%	3,53%	0,160%	0,029%	0,160%	0,029%
Difference	-4%	-3%	81%	105%	-15%	9,884%	135%	1275%

Table 11: Share of outgoing traffic observed with NetFlow and optical passive monitoring

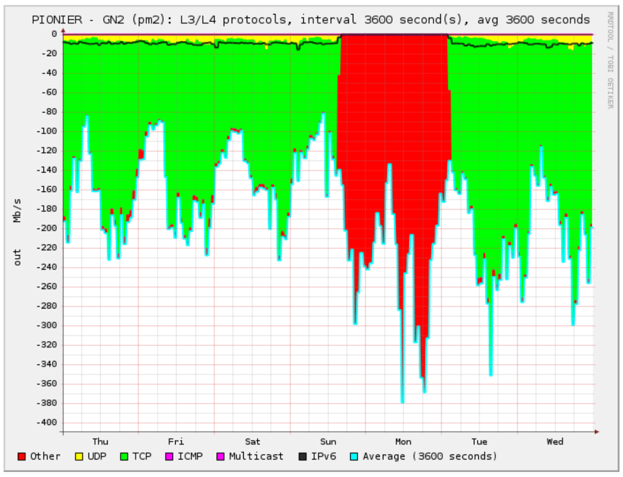
protocols are unfortunately reflected in the NetFlow analysis with a high error. Our results show that that for the other protocols observed like UDP, ICMP and IPv6, shares in the total volume of traffic vary by even more than 50%. An extreme result is for IPv6 protocol counted by NetFlow in the outgoing traffic from PIONIER (Fig. 11 where the difference is more than 10 times that the amount counted by optical passive monitoring. The understanding behind this is that the link does not carry significant amount of these types of traffic in this direction to be correctly analysed by sampling mechanism of NetFlow. This leads to high discrepancy of the observed traffic from each measurement method. Unfortunately such conclusion means that NetFlow monitoring with a sampling rate of 1/10000 we use for PIONIER NREN is not a reliable monitoring technique in short periods of time (as used in this test) where the volume of traffic is low but rather to longer trends. In such use cases optical passive monitoring fits better allowing users accurate observation of details of low volume traffic and short-time network behaviour. We also conside extending PIONIER core routers with dedicated PICs for unsampled NetFlow.

4.5.4 Traffic periodicity

We also looked at the periodicity of the network traffic over the link to GEANT network collected with the use of optical passive monitoring. We take into account a week from 11 to 17 of June 2009 with resolution of 1 hour. Fig. 21 shows the total traffic for the observed period. We can see that the traffic follows a diurnal cycle related to daily activities of researchers. There are spikes in the incoming traffic every working day at about 14:00. They are much less visible during the weekend. Also spikes in the outgoing traffic are not so obvious. Fig. 22 shows common application protocols traffic. We can again see similar behavior although in the file sharing protocols the diurnal pattern is not so much exhibited. The outgoing traffic in Fig. 22(b) does not almost show any BitTorrent type of applications traffic which may indicate that the users of that kind of tools are not much interested in resources available in Poland through research interconnection.

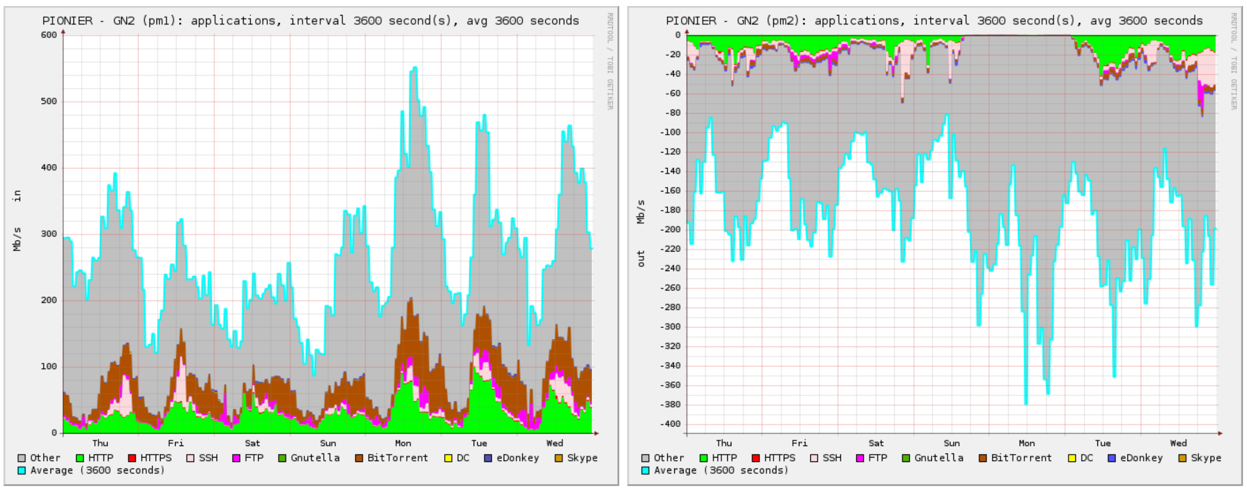


(a)



(b)

Figure 21: Diurnal cycles for total incoming to PIONIER (a) and outgoing traffic (b)



(a) (b)

Figure 22: Diurnal cycles for common application protocols in incoming to PIONIER (a) and outgoing traffic (b)

4.6 Design of a Stream-based IP Flow Record Query Language

4.6.1 Introduction

The NetFlow protocol [12], originally designed by Cisco Systems, enables routers to export summary information about the traffic flows that traverse a router. Inspired by Cisco's early work, the IETF created a standard IP flow information export protocol called IPFIX [25] A network flow is defined as an unidirectional sequence of packets between given source and destination endpoints. Specifically, a flow is usually identified by the combination of the following seven key fields: source and destination IP address, source and destination port number, IP protocol type (TCP, UDP, etc.), ToS byte, and the input interface (ifIndex). In addition to the key fields, a flow record contains other accounting fields such as packet and byte counts, input and output interfaces, bit-wise logical or of TCP flags, timestamps, MPLS labels etc. Network elements (routers and switches) gather flow data and export it to collectors for analysis.

The flow records exported via NetFlow/IPFIX provide a summary about the traffic traversing a specific router. However, raw collections of flow records still contain too many details for network administrators and they are not useful unless processed by network analysis tools. Most of the existent flow record processing tools provide mechanisms for searching of specific flows through some simple operations like filtering by an IP address or port number or generating Top-N talkers reports. However, in order to match more complex flow patterns against collections of flow records, one needs a useful flow record query language.

Given the large number of flow records collected on high-speed networks, it is necessary to reduce their number to a comprehensible scale using filtering and aggregation mechanisms. Each flow or aggregated flow has a set of properties attached to it that characterize the flow. It is to be expected that flows that correspond to similar network activities (certain applications or certain attacks) have similar properties. In addition to the properties recorded in flow records, one can derive further properties that are even more suitable to characterize the behavior of a flows. One objective when investigating traces is to detect traffic regularities such as repeating patterns, which can be associated with the usage of common network services. This approach can be further extended to detect traffic irregularities such as network anomalies or attacks, which also generate specific patterns. These patterns typically spread over several flows. For example, if an intensity peak in flow X always occurs after an intensity peak in flow Y with a fixed delay, they form a pattern describing a certain network behavior. The goal of network administrators is to detect such patterns of correlated flows.

For example, one would be interested in finding out where, when, and how often a certain Internet service is used. A concrete scenario is a network administrator who wants to detect VoIP applications by finding STUN flows generated by VoIP applications when they try to discover whether they are located behind a Network Address Translator (NAT). If one knew the pattern that is created when a service is trying to establish a connection, one could search for this specific pattern in the selected flows. We are aware that although the presence/absence of a certain pattern may be a hint for the presence/absence of a particular service this by no means proves that the service is really running/missing.

In this work we propose a flow record query language, which allows to describe patterns in a declarative and easy to understand way. This work is a followup of the early work [117], where we discussed in some detail the motivation of this research. The proposed language is able to define filter expressions (needed to select relevant flows) and relationships (needed to relate selected flows). It allows to express causal dependencies between flows as well as timing and concurrency constraints. Existent query languages as discussed in Section 6.7 are not suitable for detecting complex traffic patterns because of either performance issues (SQL-based query languages) [118, 119] or because they lack a time and concurrency dimension (BPF expressions and the other query languages we discuss). Furthermore, the new query language provides support for network specific aggregation functions, such as IP address prefix aggregation, IP address suffix aggregation, port number range aggregations, etc. which are not part of many query languages. Using the new query language, we built a knowledge base of flow fingerprints that belong to some common network services, applications and attacks. As an example, we describe the query detecting the flow fingerprint of the Blaster.A worm.

The rest of the section is structured as follows. Section 4.6.2 provides a short survey of existing flow filtering and query languages. In Section 4.6.3 we present our stream-based flow query language and in Section 4.6.4 we show an application example by using it to describe a common network traffic pattern. We conclude in Section 4.6.5 with a few remarks on ongoing work.

4.6.2 Related Work

According to [117] existing flow record query languages can be split into SQL-based query languages, filtering languages, and procedural languages.

SQL-based Query Languages

Many of the early implementations of network analysis tools used a Relational Database Management System (RDBMS) to store the data contained in flow records and therefore they use SQL-based query languages for retrieving flows. B. Nickless [120] describes a system which uses standard MySQL and Oracle DBMS for storing the attributes of NetFlow records. Using powerful SQL queries, the tool was able to provide good support for basic intrusion detection and usage statistics. With the advance of high-speed links, however, network managers could not rely on pure DBMS anymore because of performance issues. There was also a semantic mismatch between the traffic analysis operations and the operations supported by the commercial DBMS. The data used by network analysis applications can be best modeled as transient data streams as opposed to the persistent relational data model used by traditional DBMS. It is recognized that continuous queries, approximation and adaptivity are some key features that are common for such stream applications. However, none of these is supported by standard relational DBMS. Based on these requirements B. Babcock et al. [119] propose the design of a Data Stream Management System (DSMS). Together with the model the authors also extend the SQL query language so that the DSMS can be queried over time and provide examples of network traffic reports that are generated based on flow data that is stored in such a DSMS. *Gigascope* [121] is another stream database for network monitoring applications. It uses GSQL for query and filtering, which is yet another modification of the SQL query language adopted in a way so that time windows can be defined inside the query. *Tribeca* [118] is another extensible, stream-oriented DBMS designed to support network traffic analysis. It is optimized to analyze streams coming from the network in real time as well as offline traces. It defines its own stream-based query language which supports operations such as projection, selection, aggregation, multiplexing and demultiplexing of streams based on stream attributes. The query language also defines a windowing mechanism to select a timeframe for the analysis.

Filtering Languages

The *Berkeley Packet Filter (BPF)* [122] allows users to construct simple expressions for filtering network traces by IP address, port number, protocol etc. and translates these expressions into small programs executed by a generic packet filtering engine. One popular use of the BPF is in the `tcpdump` utility. The BPF rules for constructing filter expressions are also used in `nfdump` [123], which is a powerful and fast filter engine used to analyze network flow records. `nfdump` is currently one of the *de facto* standard tools for analyzing NetFlow data and generating reports. BPF expressions are also used in the *CoralReef* network analysis tool described in [124, 125] in order to generate traffic reports from collected trace files. The *Time Machine* tool described in [126] uses BPF expressions to define classes of traffic and BPF is also part of the query language used by the tool for retrieval of interesting traffic.

The `flow-tools` package [127] is another widely-used collection of applications for collecting and analyzing NetFlow data. Two of the flow-tools applications are responsible for filtering flows and generating reports: `flow-filter` and `flow-report`. The former application uses the Cisco Access Control List (ACL) format to specify a filter for IP addresses and command line arguments for specifying other filtering parameters such as port numbers, ASes etc. The latter uses a configuration file where reports can be defined by using a number of primitives.

Procedural Languages

FlowScan described in [128] is a collection of perl scripts which glues together a flow-collection engine such as the `flow-capture` application from `flow-tools`, a high performance RRD database, which is specifically designed for time series data [129], and a visualization tool. FlowScan has the capability of generating powerful high-level traffic reports, which might help operators to detect interesting traffic patterns. However, reports must be specified as separate perl modules, which is not trivial and might involve some heavy scripting.

C.Estan et al. [130] proposes an approach for detecting high-level traffic patterns by aggregating NetFlow records in clusters based on flow record attributes. Aggregation on several flow attributes results in a multidimensional cluster. Initially all possible multidimensional clusters are constructed. Then an algorithm is executed which selects only clusters that are interesting to the network administrator. It aims at retaining clusters with the least degree of aggregation (so that a bigger number of flow attributes is contained). Interesting activities are considered to be exceeding a certain threshold of traffic volume of a cluster or significant change of the traffic volume inside the cluster. Finally, all clusters are prioritized by being tagged with a degree of *unexpectedness* and presented to the network administrator as a traffic report.

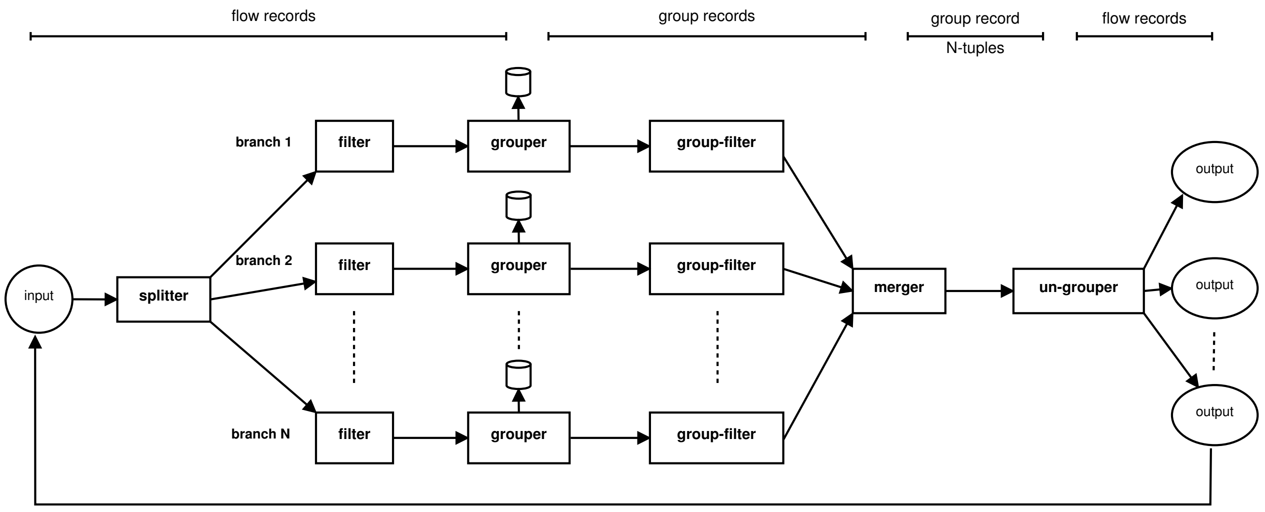


Figure 23: IP flow filtering framework

The *SiLK* Analysis Suite [131] is another script-based collection of command-line tools for querying NetFlow data. It provides its own primitives for defining filtering expressions. Unlike other network analysis tools, *SiLK* contains two applications that allow an analyst to label a set of flows sharing common attributes with an identifier. The `rwgroup` tool walks through a file of flow records and groups records that have common attributes, such as source/destination IP pairs. This tool allows an analyst to group together all flows in a long lived session such as a FTP connection. `rwmatch` creates matched groups, which consist of an initial record (a query) followed by one or more responses. Its most basic use is to group records into both sides of a bidirectional session, such as a HTTP request.

4.6.3 Stream-based Flow Query Language

Our framework for IP flow filtering follows a stream-oriented approach — it consists of a number of processing elements or operators, which are connected with each other via pipes. Each element receives an input stream, performs some sort of operation on it (filtering, aggregation etc.) and the output stream is piped to the next element. Figure 23 shows the framework and in the following sections we describe each of its elements. A complete definition of the syntax and the semantics of the elements can be found in [132]. Section 4.6.4 provides an example illustrating the usage of the primitives of the stream-based flow query language. The names of the filtering primitives in our language are closely linked to the flow record attributes in RFC 5102 [133].

Splitter

The `splitter` is the simplest operator in the IP flow filtering framework. It takes the input stream of flow records and copies them on each output stream without performing any changes on them. There is one input branch and several output branches for a `splitter`.

Filter

The `filter` operator takes a stream of flow records as input and copies to its output stream only the flow records that match the filtering rules. The flow records, which do not match the filtering rules are dropped. The `filter` operator performs *absolute* filtering, it compares the flow attributes of the input flow records with absolute values (or a range of absolute values). It can also perform comparison between the various fields of a single flow record, that is it can compare one field of a flow record against another field of the same flow record (for example source port number with destination port number). The `filter` operator does not support *relative* filtering between fields from different flow records i.e., it does not perform comparison between the flow attributes of different incoming flow records.

Grouper

The `grouper` operator takes a stream of flow records as input and partitions them into groups and subgroups following grouping rules. The grouping rules themselves are organized into rule modules, where each rule module contains a number of rules logically linked by an implicit logical and. The different rule modules on the other hand are logically linked by an implicit logical or. The rules reflect some relative dependencies and patterns among the attributes of the input flow records. The `grouper` tags each flow record with a group label and each group consists of flow records tagged with the same group label. Internally, each group consists of several not necessarily non-overlapping subgroups, which correspond to the different rule modules. The `grouper` also tags each flow record with a rule module identifier (also called a subgroup label) if the flow record satisfies the set of rules within the corresponding rule module. In order to be added to a group a flow record must satisfy the rules from at least one rule module. In case a flow record satisfies the rules from several rule modules, it is tagged with the rule module identifier of all matching rule modules and thus belongs to several subgroups. The way group and subgroup labels are stored into flow records is implementation specific, for example the `SiLK` tool [131] stores the labels in the `next-hop` field of the flow records.

For each group of flow records a group record is created. It may consists of the following attributes:

- Flow record attributes according to which the grouping was performed. This is usually a set of attributes that are unique for a subgroup and form a *key* for that subgroup. If there is a single rule module within a `grouper` definition then there will be a single subgroup and these flow record attributes will be unique for the group as well.

- Aggregated values for other flow record attributes from a single subgroup. If the subgroups within a single group are identified by `g1`, `g2` etc. then the group record may contain members such as `g1.sum(attr1)`, `g1.min(attr2)`, `g2.max(attr3)` etc.
- Aggregated values for other flow record attributes from the whole group. In such a case the aggregation is performed over the flow records of the whole group (as opposed to aggregation over a single subgroup). For example, the group record may contain members such as `sum(attr1)`, `min(attr2)`, `max(attr3)` etc. Note that in this case we can drop the subgroup label.

Once the group record is created the subgroup labels are not needed anymore and can be deleted. Finally, the group records are copied over the output stream. During the grouping operation some information from the original flow record trace is lost because of the aggregation operation during the creation of the group records. Therefore, after the grouping and tagging is performed and before the aggregation phase, the tagged flow records are copied to a temporary storage so that they can be later retrieved by the `ungroup` operator. The details of the algorithm are described in [132].

Group-filter

The `group-filter` operator takes a stream of group records as input and copies to its output stream only those group records that match the filtering rules. The group records, which do not match the filtering rules are dropped. The `group-filter` operator performs *absolute* filtering on the flow record attributes or the aggregated flow record attributes contained within the group records. It compares the flow record attributes (aggregated flow record attributes) of the input group records with absolute values (or a range of absolute values). It can also compare various group record attributes within the same group record. It does not support *relative* filtering i.e., it does not perform comparison between the flow record attributes (aggregated flow record attributes) of different incoming group records.

Merger

The `merger` operator takes several streams of group records as input and copies to its output tuples of group records that satisfy some pre-defined merging rules. The merging rules are organized in merging rule modules. Each rule module specifies a set of input branches from all branches that meet at the `merger` and a number of rules, which use group record attributes to define certain relationships among the various flow groups. If there are N input branches as input to a specific merging rule module, the output stream of that rule module will consist of N-tuples of group records, one group record per input branch. The output stream of one of the merging rule modules is the output of the whole `merger` operator. There is always exactly one rule module that produces the output stream for the `merger` operator and that rule module must be the first one defined in the `merger` definition.

In most cases there will be only one merging rule module and the tuples of group records that it generates will produce the `merger` output stream. Using one merging rule module allows us to define the existence of certain patterns among the various flow groups. However, in order to be able to check for patterns that do not exist we will need more than one merging rule module. For example consider the following two queries:

Q1 Find the flow records that correspond to a TCP connection between source IP address A and destination IP address B, port `ftp`, followed by a TCP connection between source IP address B, port `ftp-data` and destination IP address A.

Q2 Find the flow records that correspond to a TCP connection between source IP address A and destination IP address B, port `ftp`, followed by a TCP connection between source IP address B, port `ftp-data` and destination IP address A only if these two connections are not preceded by a TCP connection between source IP address A and destination IP address B, port `http`.

In more straightforward words, the first query aims at detecting a FTP file transfer between A and B, while the second query aims at detecting a FTP file transfer between A and B only if A did not already download the respective file using HTTP. While query *Q1* is relatively easy to define using a single rule module, for query *Q2* we will need a more complicated mechanism to check for the condition "A HTTP transfer did not already take place between these two entities".

In such a scenario we first perform the merging using the module rules that produce the `merger` output stream. Before copying the resulting tuples to the output stream, however, we feed them into the other merging modules and for each such tuple we check if the corresponding merging module would produce some output. These additional merging rule modules are used to define certain patterns that should not be observed and therefore a resulting tuple is only copied to the `merger` output stream if it does not generate any result when fed into any of the additional merging rules modules.

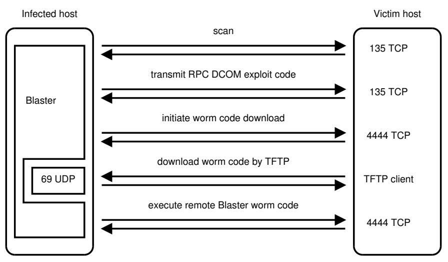
In general, the `merger` operator allows to perform grouping at a more general level compared to the `grouper` operator. Another powerful feature of the `merger` operator is its capability to express timing and concurrency constraints among various traffic groups by using Allen's time interval algebra [134].

Ungrouper

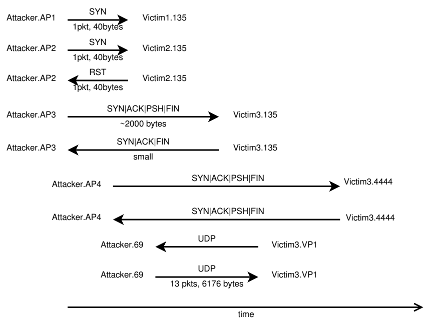
The `ungrouper` operator takes a stream of group record tuples as an input. For each group record tuple it expands the flow groups contained in the tuple using the labels and the flows stored in the temporary storage during the grouping phase of the `grouper` operator. Finally, for each group record tuple it outputs a separate stream of flow records. The flow records obtained from each group record tuple are ordered by their timestamps and presented to the viewer in capture order. Any flow record repetitions are eliminated, that is if a flow record is part of several flow groups within the group record tuple it is shown to the viewer only once. Each output stream is considered to be a result/match of the query operation performed by using the IP flow filtering framework. A query operation may return any number of results or no results at all and should clearly distinguish the flow records that belong to different results.

4.6.4 Application

In this section we present the traffic pattern generated by a computer infected with the Blaster.A worm and define this pattern using our stream-based flow record query language. The Blaster.A worm [135] is a recent Internet worm, which exploits the Microsoft


Infected host
Victim host
Blaster
69 UDP
scan
transmit RPC DCOM exploit code
initiate worm code download
download worm code by TFTP
execute remote Blaster worm code
135 TCP
135 TCP
4444 TCP
TFTP client
4444 TCP

(a) Packet level breakdown



(b) Flow level breakdown

Figure 24: Packet level and flow level breakdown of a Blaster infection

Windows Remote Procedure Call DCOM vulnerability. Upon successful execution, the worm attempts to retrieve a copy of the file `msblast.exe` from the compromising host. Once this file is retrieved, the compromised system then runs it and begins scanning for other vulnerable systems to compromise in the same manner. Dübendorfer et al. [136] describe the various stages of the Blaster worm infection and perform an in-depth packet and flow level analysis.

The characteristic network activity (an infected attacker trying to infect other hosts on the network) associated with such an attack consists of the steps described in Figure 24(a). The flow-level breakdown of the Blaster attack is shown in Figure 24(b). The flow record fingerprint of a Blaster infection consists of the following sequence of flows (order is important):

- More than 20 flows originating from the attacker directed to port 135 of different hosts. These flows are small since they only carry a SYN packet. This represents the scanning activity of the attacker. Some of these scans may trigger a reverse flow consisting of RST packets.
- In a successful attack there will be a pair of bigger flows (longer and carrying more data) to and from port 135/TCP of the victim
- The pair of flows representing the TCP connection to port 135 of the victim is followed by a pair of flows representing the TCP connection to port 4444.
- The next step is a pair of flows to and from port 69/UDP of the attacker representing the TFTP transfer of `msblast.exe`. The flow to the attacker slightly precedes the flow from the attacker since the connection is initiated by the infected host. These two flows end before the flows representing the TCP connection on port 4444 from the previous step

In order to describe a Blaster worm infection in our IP flow filtering framework we use the definitions below. We define one branch for each Blaster stage as specified in the flow fingerprint.

The first branch detects the scanning activity performed by the attacker. Initially, the `f_scan` filter picks out the flow records that have a destination port 135/TCP. Then the grouper g_scan, which consists of a single group module $g1$, partitions the flow records into groups, which have the same source IP address and the destination IP addresses consist of a block of subsequent IP addresses.

```
splitter S{}

filter f_scan {
   dstport = 135
   proto = tcp
}

grouper g_scan {
   module g1 {
      srcip = srcip
      dstip = dstip relative-delta 1
      stime = stime relative-delta 5ms
      stime = stime absolute-delta 5s
   }
   aggregate srcip, union(dstip),
             min(stime) as stime,
             max(etime) as etime,
             count
}

group-filter gf_scan {
   count > 20
}

filter f_victim {
   srcport = 135 OR dstport = 135
   proto = TCP
}

grouper g_group_tcp {
   module g1 {
      srcip = dstip
      dstip = srcip
      srcport = dstport
      dstport = srcport
      stime = stime relative-delta 5ms
   }
   module g2 {
      srcip = srcip
      dstip = dstip
      srcport = srcport
      dstport = dstport
      stime = stime relative-delta 5ms
   }
   aggregate g1.srcip as srcip,
             g1.dstip as dstip,
             min(stime) as stime,
             max(etime) as etime
}

filter f_control {
   srcport = 4444 OR dstport = 4444
   proto = tcp
}
```

```
filter f_tftp {
   srcport = 69 OR dstport = 69
   proto = udp
}

grouper g_tftp {
   module g1 {
      srcip = dstip
      dstip = srcip
      srcport = dstport
      dstport = srcport
      stime = stime relative-delta 5ms
   }
   module g2 {
      srcip = srcip
      dstip = dstip
      srcport = srcport
      dstport = dstport
      stime = stime relative-delta 5ms
   }
   aggregate g1.srcip as srcip,
             g1.dstip as dstip,
             min(stime) as stime,
             max(etime) as etime,
             g2.sum(bytes) as bytes
}

group-filter gf_tftp {
   bytes > 6K
}

merger M {
   module m1 {
      branches A,B,C,D
      A.srcip = B.srcip
      A.srcip = C.srcip
      A.srcip = D.dstip
      B.dstip = C.dstip
      B.dstip = D.srcip
      B.dstip in union(A.dstip)
      A < B OR A m B OR A o B
      B o C
      D d C
   }
   export m1
}

ungrouper U{}

input -> S
S branch A -> f_scan -> g_scan -> gf_scan -> M
S branch B -> f_victim -> g_group_tcp -> M
S branch C -> f_control -> g_group_tcp -> M
S branch D -> f_tftp -> g_tftp -> gf_tftp -> M
M -> U -> output
```

Additional constraints ensure that a scanning attack lasts at most 5s and each scanning attack should not be interrupted for more than 5ms (since attackers usually generate the small SYN packets at a very high rate). Finally, for each group the grouper operator retains the source IP address, the set of destination IP addresses, the start and end time of the attack as well as the number of flows in the group. Each such flow group now contains information about the scan attack performed by a single host. The newly created group

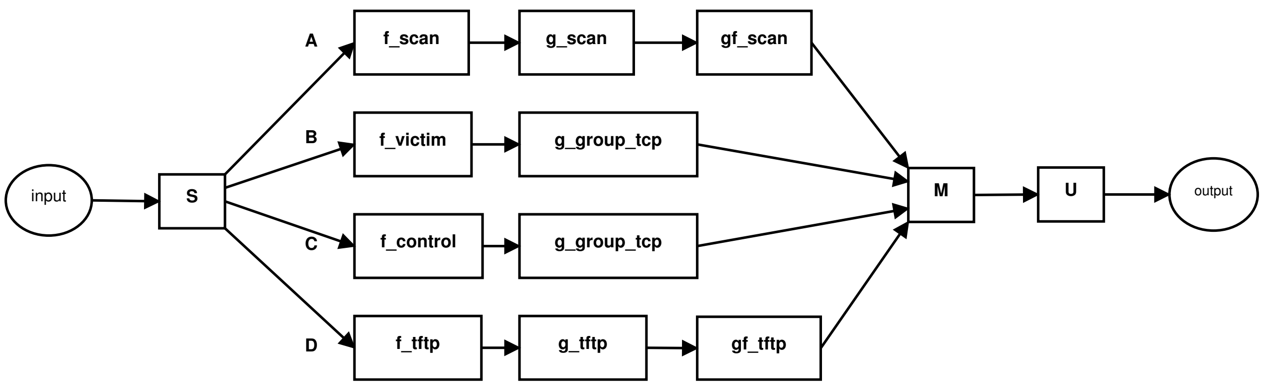


Figure 25: Capturing Blaster worm infections with the IP flow filtering framework

records are passed to the group-filter operator, which filters only these group records that refer to flow groups containing more than 20 flows. That is, we consider the traffic from a flow group a scanning attack only if the attacker has scanned more than 20 hosts. If the attacker has scanned less than 20 hosts we consider the flow group normal traffic activity and drop it.

The second branch, which consists of the filter `f_victim` and the grouper `g_group_tcp` aims at capturing the TCP connection that the attacker established with the victim i.e., a TCP connection between the attacker and port 135 of the victim. The filter picks out flows with a source or destination port 135. The grouper `g_group_tcp` then aggregates all flow records that correspond to the same TCP connection in one group. The group module $g2$ adds to the flow group all flow records that have the same source and destination transport endpoints as the flow record that generated the flow group. The group module $g1$ adds the flow records that correspond to the opposite direction of the TCP connection i.e., it adds those flow records for which the destination transport endpoint is the same as the source transport endpoint of the flow record that generated the group and vice versa (the source transport endpoint is the same as the destination transport endpoint of the flow record that generated the group). For each TCP connection then `g_group_tcp` retains the source and destination IP addresses as well as the start and end times for each group.

The third branch aims at identifying the control connection between the attacker and port 4444/TCP of the victim. The filter `f_control` picks out the flow records with a source or destination port number 4444/TCP and the grouper `g_group_tcp` partitions them into groups as already explained.

The last branch aims at capturing the TFTP download, which gets initiated by the victim host. The filter `f_tftp` picks out the flow records, which belong to UDP traffic to or from port 69 (tftp). Then the grouper `g_tftp` is very much like `g_group_tcp` in terms of the partitioning that it is performing. Both groupers contain two group modules, which aim at detecting the forward and backward direction of each TCP connection / UDP transfer. `g_group_tcp` partitions the incoming stream of TCP flow records into groups so that each group corresponds to a separate TCP connection and `g_tftp` partitions the incoming stream of UDP flow records into groups so that each group corresponds to a separate TFTP/UDP transfer (to the extent to which we are able to distinguish different UDP/TFTP transfers by using the `delta` and `relative-delta` parameters). In general the grouper is not aware of what the incoming stream of flow records contains, thus it is not aware if it is grouping TCP or

UDP flow records. Therefore, `g_group_tcp` can already do the job of splitting the incoming flow records into groups where each group represents a separate UDP/TFTP transfer. In this case however, we are also interested to retain the amount of data exchanged in each UDP/TFTP transfer in order to do some filtering later in the `group-filter`. Therefore, the specification and the interpretation of `g_tftp` is the same as the one of `g_group_tcp` i.e., the group module $g2$ adds to the flow group all flow records that belong to the forward direction of the UDP/TFTP transfer (as compared to the first flow records that generated the group) and the group module $g1$ adds all flow records that correspond to the backward direction of the UDP/TFTP transfer. The only addition of `g_tftp` as compared to `g_group_tcp` is that the former also retains the amount of data exchanged within each group (that is within each TFTP transfer).The resulting group records are then passed to a group-filter which retains only these group records, which represent a TFTP exchange of at least 6K since the netflow fingerprint of this stage specifies that the virus is approx. 6176 bytes.

The next step consists of defining the merger M and the merging conditions. M contains a single merging rule module $m1$, which takes the four already defined branches A, B, C and D as an input. The first three rules from the merging rule module definition specify that the source IP address of the attacker should be the same during the different stages of the Blaster infection (since we want to retrieve a single attack from a single attacker host to a single victim host). The next three rules specify that the IP address of the victim should also stay the same and that the victim should be one of the scanned hosts from the first stage of the Blaster infection. The last three rules express the time and concurrency constraints among the four branches using Allen's time interval algebra. We assume that the scanning phase takes place completely `before`, `meets` or `overlaps` with the stage of successful TCP connection establishment on port 135 with the victim. Furthermore, the connection on port 135/TCP `overlaps` with the control connection on port 4444/TCP. Finally from Figure 24(b) one notices that the TFTP transfer happens `during` the control connection (i.e., the TFTP transfer is entirely contained within the control connection).

The last part of our definition consists of defining the ungrouper and linking the already defined elements using pipes to build a model for our IP flow filtering framework.

4.6.5 Conclusions

After a careful analysis of the pros and cons of the existing filtering and query languages [117], we designed a new IP flow filtering framework and an associated filtering language. The language primitives were chosen in such a way that the new IP flow record query language has the capability to describe aggregation and comparison based on flow record attributes. In addition various dependencies such as flow correlation, timing and concurrency constraints, flow ordering and causal relationships can be expressed. The IP flow filtering framework has a limited number of operators, which can be defined and linked in a very flexible manner. This makes it relatively straightforward to use for the definition and detection of various traffic patterns.

In order to evaluate our new IP flow record query language, we collected a set of traffic patterns that belong to some popular network applications and services [132]. We analyzed HTTP and FTP transfers, the propagation of some well-known worms as well as

Skype traffic. The flow fingerprint of each traffic pattern was derived and written down using the IP flow record query language.

We are currently implementing a prototype consisting of a parser, which reads the flow pattern definition, and an execution engine, which implements the operators or our IP flow filtering framework. For some of the more complex operators such as the grouper and the merger, there will be a need to do some research in order to decide which algorithms and heuristics should be used in order to optimize the performance. In addition, one should consider various possibilities for flow storage and choose the most efficient one. In the future, we envision that protocol experts will, assisted by an interactive flow visualization system, develop queries for specific scenarios. Once our implementation is complete, we can test these queries and share them with non-experts through a knowledge base so that they can be easily matched against flow traces.

4.7 Changing Network Behavior

4.7.1 Introduction

Traditional Intrusion Detection Systems (IDS) analyse the network traffic for the appearance of patterns stored in a database. While one can achive high and reliable detection rates, two important problems remain: First, only already known attack patterns can be found which are available in the database, thus those systems are hardly able to detect new and unknown attacks. There always will be a delay until new patterns are available. Second, analysing the whole payload is computationally intensive for high bandwidth. Sharing out the examination of the traffic into multiple parallel systems is possible but lead into expensive systems. Even more, the ability to detect distributed attacks can be lost by bad partitioning of the traffic. Therefore, a lot of systems only analyse the headers of the network packets. For a better application of the divulging high speed network connections, miscellaneous research focus on the enhancement of the detection capabilities in this environments with link speeds about 10 Gbps. *Machine Learning Intrusion Detection* is an IDS developed by Fraunhofer that uses machine learning approaches to detect new and unknown attacks. In the subsequent project called ReMIND [137], the research focus on adapting the MIND technology under realtime aspects and to handle high data transfer rates. Other approaches use Field Programmable Gate Arrays thus utilising the possibilities for a high parallelisation of the payload analysis while using low-cost hardware[138].

Neural networks are capable to enhance the detection and interpretation capability. Due to their ability to find behavioral patterns in the attacks which could be learned, detection of yet unknown attacks is possible [139]. Even this is one of the main advantages of neural networks, they suffer from a bad adaptability to changing environments after the learning phase has been finished. Modular and hierarchical artificial neural networks are a possibility to set about these issues. Neural networks can be constructed in a way that they are compositional. For that, the complexity of the whole neural network can be divided into multiple single parts. Berg and Spaanenburg showed how to realise compositionality under further conditions [140]. Potter proposed a *Learning Intrusion Detection System* based on a blackboard architecture [139] for gaining online-learning capabilities.

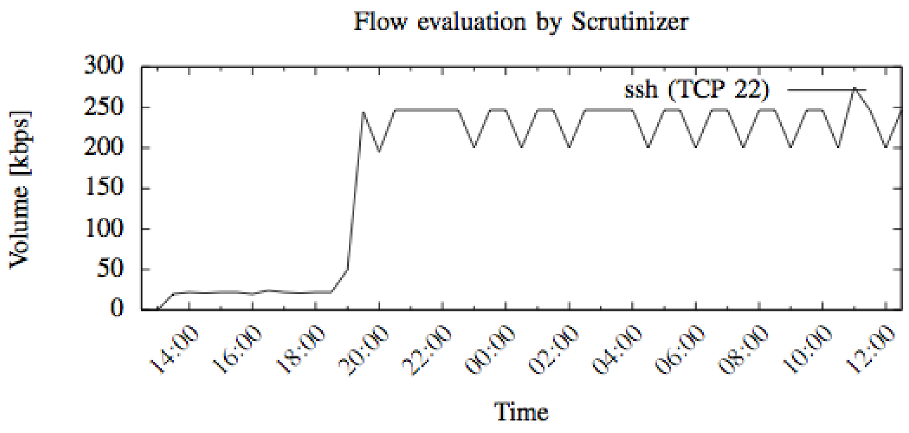


Figure 26: Massive injection of NetFlow packets.

The concepts of IDSs are extended when the network infrastructure and its traffic is examined as a whole. The correlation of the available traffic data and alerts is difficult to handle and a fundamental precondition to detect malicious scenes. *Network Security Situation Analyses* considers all information about the network, and is able to project the next behavior of it [141]. Situational factors can be extracted and evaluated by their dependencies to get the security situation of the network [142]. For example, a special type of intrusion detected by an IDS sensor could be such a factor.

By statistical analyses of the data transported through the network, knowledge about the services and the transmitted data can be gathered. NBA based IDSs belong to very efficient systems. There, anomalies are detected by the evaluation of the behavior of the network traffic (e.g. [143]). Possibilities for gathering the needed information about the transmitted data are capturing the raw data or using flow information generated by routers and switches. Therefore, one of the main pros of NBA is that no host sensors have to be installed and the system can be integrated in an existing environment easily.

In a flow-capable device, a flow generator analyses the traffic stream and builds data packages with statistical information about it. Based on the used type of flow, these packets contain a different amount of information about the network connections, for example the source and destination IP-addresses and -ports, the amount and type of data transmitted for a connection or the amount of data transmitted by a particular port of a switch. Therefore, a stream of packets with flow information is sent, whereby the number of packets depends on the type of flow and the configuration of the involved devices. The resulting flow data can be used for monitoring the network traffic, optimising the routing and the infrastructure usage and for the calculation of QoS parameters as well as the detection of attacks.

Two important standards for flow data are *NetFlow* invented by Cisco and *sFlow* mainly based on the work of Hewlett Packard and the University of Geneva. NetFlow generates statistics of the network traffic by counting every packet and extracting the IP header as well as the TCP or UDP header. Every packet arriving at a NetFlow-enabled port is categorized by seven key fields. Packets with the same categorisation are assigned to the same flow. After a conversation has been finished, the periodic flush is reached or flow tables are going to become full, a NetFlow packet is generated and sent to the configured flow destination. Hereby, a single NetFlow packet can contain information about up to 20 or 30 conversations.

The initial version of NetFlow was developed by Cisco in 1996. These capabilities are embedded in the Cisco Internetwork Operating System (IOS). The original definition was extended several times, latest release was version 9 in 2004 (RFC 3954 [12]). Still, furthest widespread is version 5 [144]. NetFlow version 9 is also the base for IPFIX (RFC 3917 [13], RFC 5101 [25]) which is developed by the Internet Engineering Task Force. Because of the software-implemented measurement, NetFlow can utilise considerable amounts of CPU usage when enabled on high speed ports with data rates of 1 Gbps and higher. Anyway, when configured correctly, the increase of the CPU utilisation should not be higher than about five percent, depending on the router platform and the number of flows traversing through the device [145].

The first version of sFlow was released in 2001, based on the work started with network-wide monitoring using packet sampling presented by Hewlett Packard and the University of Geneva in 1991 [146]. Since 2002, hardware devices with sFlow capability are available for example from Hewlett Packard, Allied Telesis and D-Link. sFlow is defined in RFC 3176 [147]. It combines interface counters, flow samples and the state of forwarding and routing table into flow packets which are sent immediately to the flow collector [148]. While NetFlow is able to include every packet passing a monitored port into the analyses, sFlow is a sample-based technology, only reckoning every n-th packet. A common value is one packet per second depending on the configuration of the sampling. Therefore, it is hardly possible to get an accurate representation of the whole traffic of an interface and algorithms are necessary to calculate a statistical representation of the total traffic. sFlow acts with consideration, using only very small amounts of bandwidth, typically about 0.1 percent [148]. While NetFlow is software based and part of Ciscos IOS, sFlow is hardware based using dedicated chips. Therefore, no additional CPU utilisation is necessary. Many features of sFlow are also available in NetFlow v9/IPFIX, but there are no push-based counters in NetFlow. By using the counter export of sFlow devices, data that is normally polled by SNMP can be retrieved very efficient.

Both types of flow, NetFlow as well as sFlow, provide a lot of valueable statistical information about the data flowing through the network and are utilisable for describing the network behavior and anomaly detection. Several companies offer NBA products based on the evaluation of flow data, e.g. Lancope StealthWatch [149], Riverbed Cascade [150] or Sourcefire RNA and 3D [151].

An important challenge of NBA systems is the necessity to accomplish an extensive learning phase wherein the typical behavior of the network environment has to be learned. Even some NBA systems are also able to detect intrusions or malicious behavior while still in the learning phase, this is at most possible in a very slow rate. There is only a very limited or none detection capability available while being in this phase. Rather important is the learning phase itself. Within a complex network topology, this procedure can last for several weeks [152], [153]. During that time, the system builds the behavior model of the environment. Obviously it is not possible to carry out the learning process in an isolated and secured environment because of the missing real-world traffic. Also the (productive) network cannot be disconnected for several weeks in most cases. Therefore, the learning process has to be carried out in the unsecure real-world environment. Assuring a correct learning phase is not a trivial problem when integrating such an IDS.

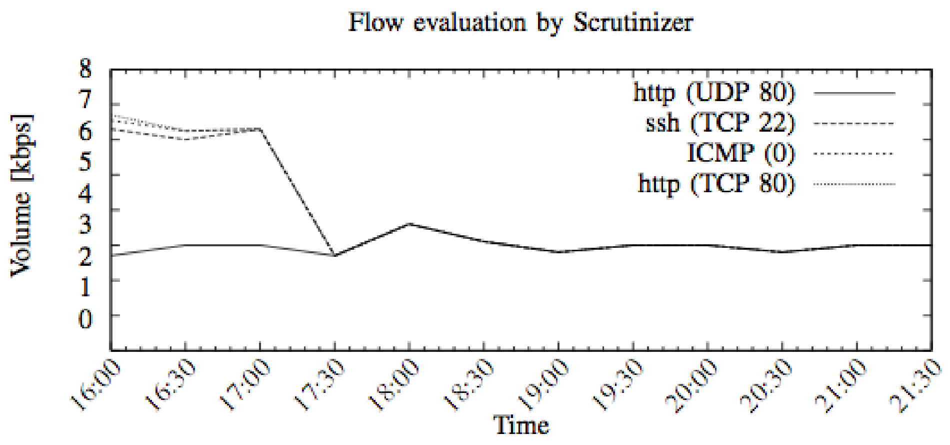


Figure 27: Manipulation of the payload of the NetFlow datagrams.

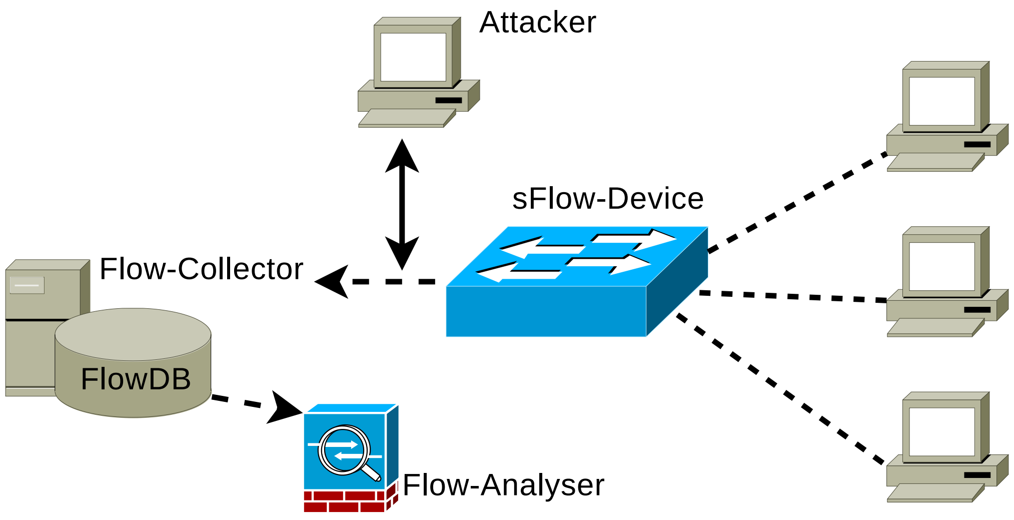


Figure 28: Attack scenario for the sFlow environment.

In the area of neural networks, few work has been done to investigate the optimisation of the construction of the network with respect to the learning phase. For realising faster learning capabilities, a problem-specific design of the neural network can be done. Moraga was able to construct neural networks for specific problem statements which were able to solve the given task completely without the need of a learning phase [154].

To overcome the threats in the learning phase of the IDS, we are going to examine the possible attack scenarios against the neural network in the learning phase in depth. With the knowledge gained by this, we will construct pre-processed modules for the composition of the artificial neural network, therefore not be endangered for manipulation during the learning phase. Depending on the communication structures and evaluation techniques, the attacker or malicious code has several options to manipulate the learning phase of a NBA system.

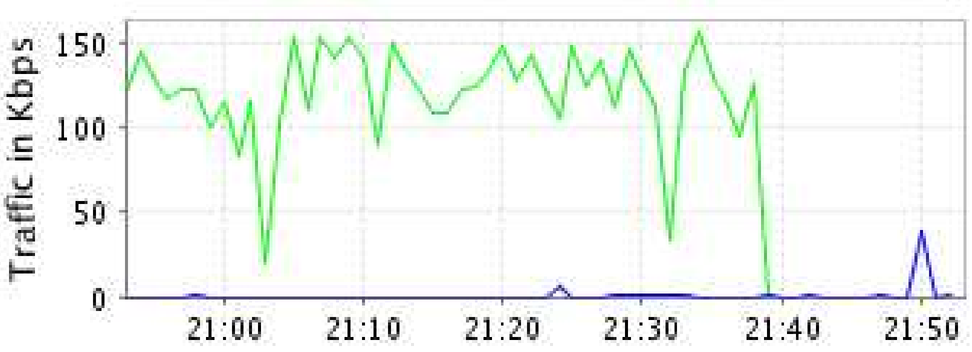


Figure 29: Example of an output graph generated by ManageEngine NetFlow Analyzer.

4.7.2 Scenario and Experimental Evaluation

In a first step, a Cisco 7200 router was used for the evaluation. Two networks had been connected via the Cisco 7200 and different network loads were transmitted between these two networks. NetFlow had been enabled, and the collection and analyses of the flow data was done by a free version of Scrutinizer v6, a NetFlow and sFlow Analyser from plixer International [155]. Multiple attack scenarios had been evaluated, beginning with an injection of flow packets based on data of existing packets sent to the collector. By sniffing the flow data and constructing the new payload based on the sequence number and times, as well as the source IP address and the MAC address of the originator thus of the Cisco 7200 router, spoofed packets had been sent by the use of the network packet injection tool nemesis [156].

Two further attacks had been carried out namely dropping original packets and manipulating their payload. Therefore, a transparent bridge had been inserted into the network connection.

Figure 26 shows the result recorded by Scrutinizer when exposed to a massive packet injection. Note that the graphs are redrawn with gnuplot [157] for achieving a higher quality. In this scenario, the original packets had been dropped while new packets generated on the basis of the dropped ones had been injected. The traffic amount represented by the NetFlow data was increased at 19:00, therefore giving a strongly dominant faked data connection. By manipulating the payload of the NetFlow packets and forwarding them afterwards, more sophisticated attacks are able to be conducted. The result of fading out the the real data streams is shown in Figure 27. The constant traffic generated through a connection on http (UDP 80) is a faked traffic stream while the amount of data transmitted by the real streams had been faded out beginning at 17:00 by reducing the values denoted by the NetFlow packets. Therefore, after 17:30 only the faked traffic is seen by the flow collector.

It was easy to manipulate the NetFlow traffic without triggering notifications and changing the evaluated behavior completely when no cryptographical procedures for the transmission of the data had been applied. Even with the usage of secured communication, several possibilities for manipulating the statistical evaluation remain. In a further scenario, we generated real traffic through the router by introducing two transparent bridges, one on each side of the router in the link to the connected networks. Based on this, every type

Type	TCP	UDP	Crypted	Effect
Packet Injection		X		Legitimate
Packet Dropping		X	X	Tamper
Packet Manipulation	X	X		Hide, Legitimate
Resource Exhaustion	X	X	X	Tamper
Traffic Generation	X	X	X	Tamper, Legitimate

Table 12: NetFlow manipulation options

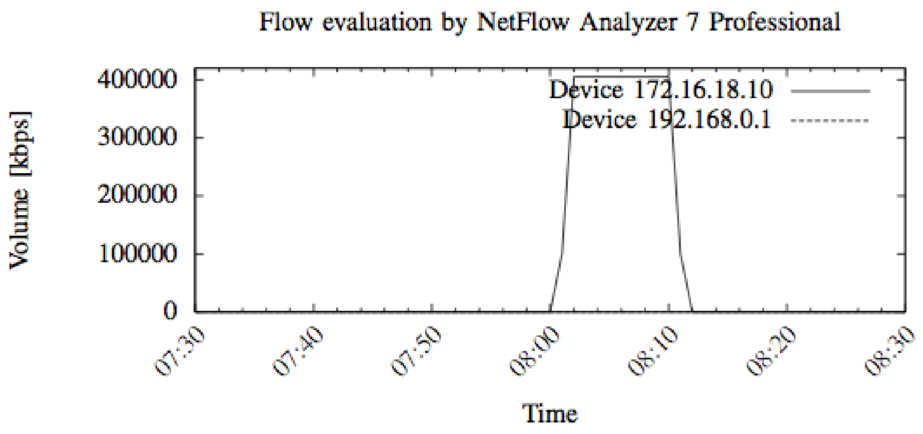


Figure 30: Massive injection of sFlow packets.

and amount of traffic can be sent through the router thus producing flow packets which are learned as normal behavior during the learning phase of the system.

Table 12 gives an overview of the possible attack scenarios against NetFlow.

sFlow is also used for the behavior evaluation by numerous NBA systems. Therefore, we set up the scenario shown in Figure 28 for manipulating the sFlow data. A Hewlett Packard ProCurve 5304xl was configured and used for the generation of the flow. The sample rate was set to produce one flow packet per second. Various work stations were connected for carrying out communication via the sFlow capable ethernet switch. For the traffic flowing through the switch, sample packets were transferred to the flow collector. This time, we used the ManageEngine NetFlow Analyzer 7 Professional Plus [158] for the collection of the flow information. A sample output of a graph generated by NetFlow Analyzer 7 is shown in Figure 29. Consecutively, outputs will be redrawn by gnuplot to improve quality.

Based on the datagram of an original flow packet of the switch, the spoofed packets had been constructed. Table 13 lists the most important entries. All values are 32 bit long until otherwise named.

sFlow packets are transmitted as UDP packets. Because of its nature of sampling the reviewed traffic, the loss of some packets is not critical and therefore typical not announced by the collector for a period of time. From address 0x00 to 0x29, ethernet-, IP- and UDP-header are in place. Beginning with address 0x32, the IP address of the agent sending the flow data is entered. The datagram sequence number is located at 0x3a, followed by

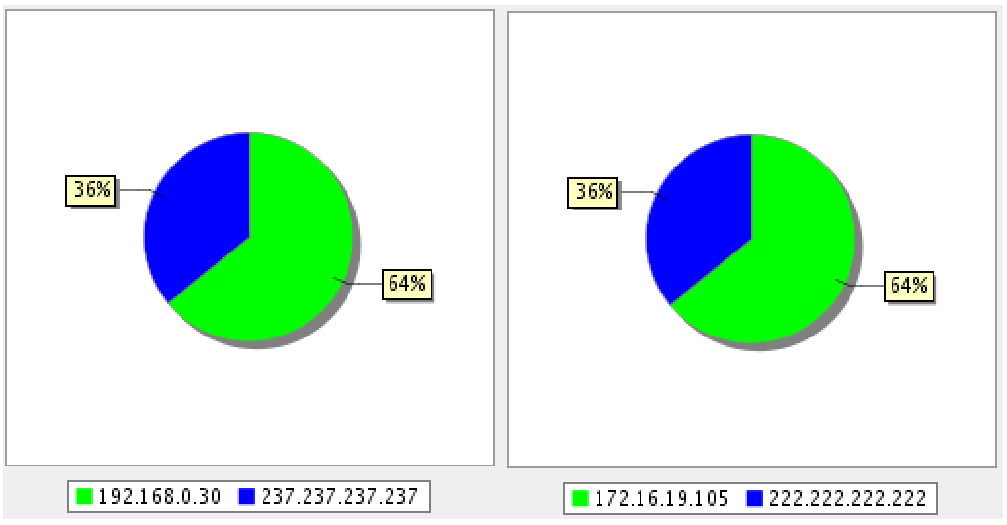


Figure 31: Connection statistic of device 172.16.18.10 at the beginning of the injection. The real traffic extinguishes the higher amount of traffic.

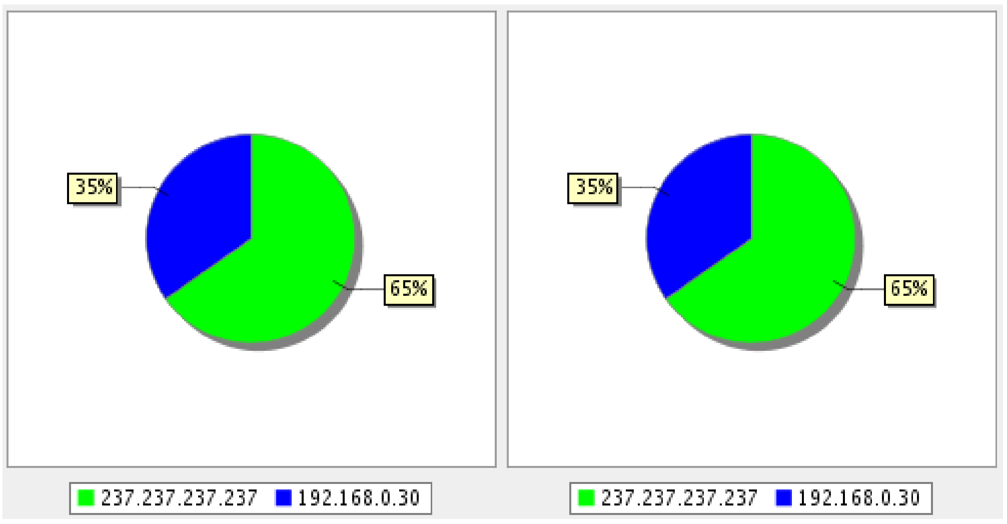


Figure 32: Connection statistics at the end of the injection. The higher amount of traffic is represented by the faked connection.

Address	Value	Meaning
0032	ac 10 13 0a	Agent address
003A	00 00 c6 17	Sequence Number
003E	0b 5f 0c b6	System Uptime
0042	00 00 00 01	Number Samples
0046	00 00 00 01	Sample Type
006A	00 00 00 01	Number Flow Rec.
006E	00 00 00 01	Flow Format
0076	00 00 00 01	header protocol
007A	00 00 05 ee	Length before Sampl.
007E	00 00 00 04	Stripped Bytes
0086	ab ab ab ab ab ab	Destination MAC
008C	ba ba ba ba ba ba	Source MAC
00A0	ed ed ed ed	Source IP
00A4	de de de de	Destination IP
00A8	00 16	Source Port
00AA	e5 26	Destination Port

Table 13: Example sFlow records

the uptime of the switch denoted in micro-seconds. Each flow datagram can consist of multiple samples, indicated by the number of samples at address 0x42. The sample type is set to the flow sample type, represented by the value `1` at address 0x46.

Every flow sample can consist of multiple flow records, announced by the value at address 0x6a. The further values depend on the used flow data type, we used a raw packet header with ethernet and IPv4 data for the injection. Therefore, the source and destination MAC addresses can be found at 0x8C respectively 0x86 while the IP addresses are at 0xa0 and 0xa4.

At first, packets were injected to feign additional network connections. In the payload of the spoofing packets, 222.222.222.222 was used as destination and 237.237.237.237 as source IP address to provide an easy recognition in the analyses. ssh-traffic was simulated, thus using port 22 (0x16, see Table 13 at address 0xa8). The written payload was injected on a regular basis with a time interval of one second using the command line tool nemesis. The resulting analysis of the flow collector is shown in Figure 30. The effect of the injection of the packets is additionally shown in Figure 32.

As one can see, in the basic configuration the color usage for plotting the graphs is based on the amount of transferred data, therefore the highest data volume is always drawn in the same color. Figure 31 shows the analysis of the flow collector to the beginning of the sFlow attack. The spoofed packets representing a ssh-connection between 237.237.237.237 and 222.222.222.222 are about 36 percent of the whole traffic running through the interface, while 64 percent are used by a real connection between 192.168.0.30 and 172.16.19.105. After commiting the attack, 65 percent of the analysed

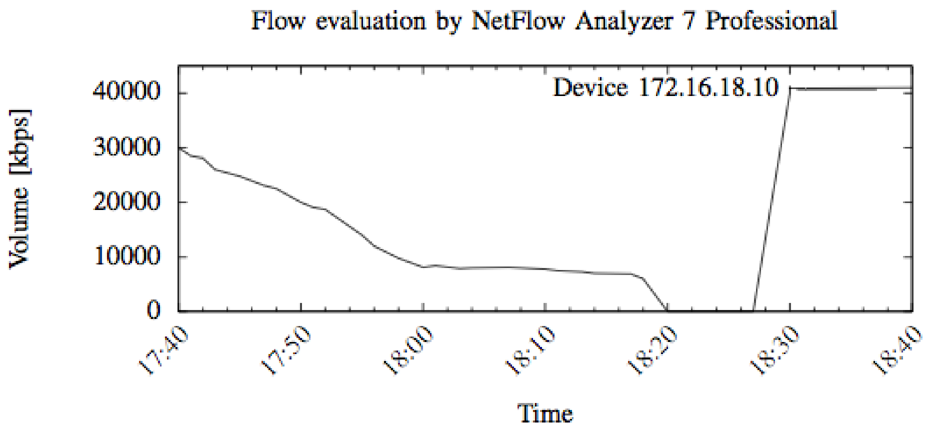


Figure 33: Manipulation of sFlow payloads.

data is from the virtual connection injected by the faked packets, while the real connection only goes into with 35 percent. Therefore, the proportion of the traffic streams has switched, hardly recognizable in the visual representation.

In that case, an administrator can overlook the change of the IP address provoked by a manipulation of the flow data easily, therefore supporting the spoofed learning phase of a NBA system.

For further affecting the learning phase, we introduce a transparent bridge between the sFlow generating switch and the flow collector. With this setup we are able to enforce two further attacks: Dropping off flow packets or the manipulaton of their payload. The leakage of packets is not critical when using sFlow, but the collector will typically present a warning when no sample packets arrive for a longer time from a flow instance. Because of the numerous variants a sFlow packet can be built, the general manipulation of incoming packets to present a special, faked type of traffic could be complicated. Therefore, a continuous stream of flow packets can be sent and because of the nature of the sampling, changes of the payload can not be detected by the collector when not cross-checked with policies or by other means. Thereby, the correct duplication of the addresses including agent id, SNMP ports etc. is from essential importance otherwise resulting in the appearance of a new flow device in the flow analyser. Figure 33 shows the graphical analyses done by the flow collector of the described attack. Note, that the NetFlow Analyzer summarizes the overall traffic of the device in this representation, but an additional breakdown to the single interfaces and their running connections is possible. The evolution of the amount of traffic was done by the manipulation of the *frame length before sampling* data (address 0x7a, see Table 13). After fading out the original traffic, new flow packets with a new faked connection are injected while the original datagrams were dropped completely. The most easy way to control the faked data volume represented by the generated sFlow packets is by changing the value of the frame length (address 0x7a in Table 13).

Table 14 gives an overview of the possible attacks against sFlow based collectors. As one can see, the types of attacks used in the NetFlow-scenario are possible to conduct in the sFlow-scenario, too. Because there are no security mechanisms in the case of sFlow, manipulation of the packets is always possible. With a careful generation of the sFlow packets, arbitrary traffic can be spoofed to the collecting process.

Type	Effect
Packet Injection	Legitimate
Packet Dropping	Tamper
Packet Manipulation	Hide, Legitimate
Traffic Generation	Tamper, Legitimate

Table 14: sFlow manipulation options

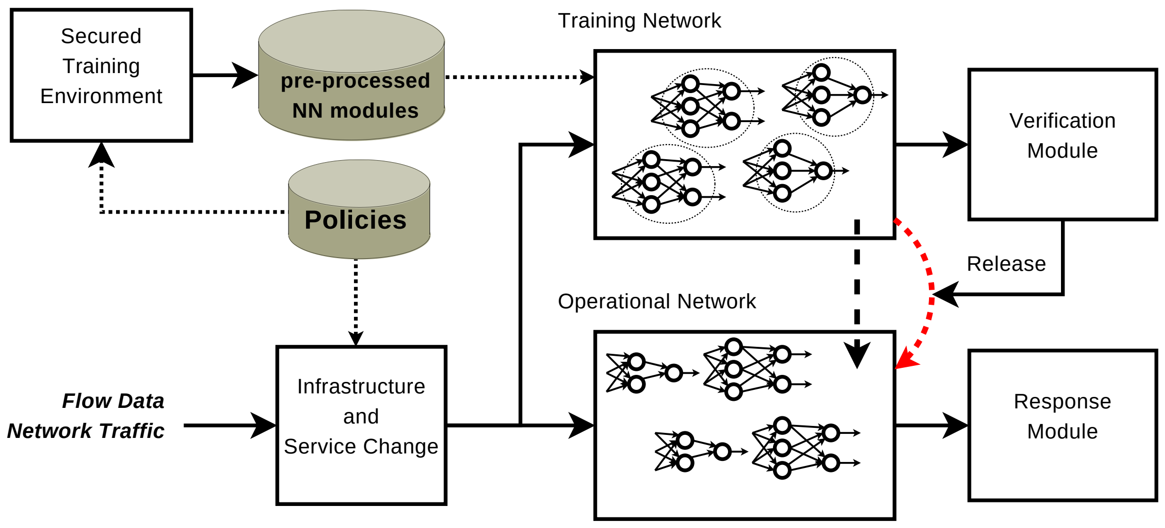


Figure 34: Hardened Fast Learning Artificial Neural Network Architecture.

Again, the usage of two transparent bridges is also possible for the injection of faked real *traffic* as it is in the NetFlow scenario. By generating traffic in the first bridge, sending it through the HP ProCurve and afterwards dropping it at the second bridge, any traffic can be presented for learning as normal network behavior. Of course, on the contrary to inject flow packets, this can produce a high amount of traffic therefore endangered to be detected.

4.7.3 Assessment and Conclusion

Protracted learning phases of NBA systems open up a high potential for danger in the first weeks of such a system.

A complete manipulation of the flow data is easily possible when the flows are not secured by further means. Especially sFlow-based data can be spoofed easily, because no specific security mechanisms are provided [159].

Also when the packets are encrypted in a NetFlow-scenario, multiple attack possibilities remain. The RFC 5101 specifies known vulnerabilities. Thus, data can be taken out by a man in the middle attack [25] when Datagram Transport Layer Security (DTLS) [160] is used for encryption. Whether TLS and DTLS can prevent the creation of fake packets, these protocols can be attacked by resources exhaustion. The possibility to manipulate the statistics by injecting whole data streams through the analysing devices is either way available. Monitioring the IP addresses is not sufficient when appropriate addresses are faked, ARP spoofing techniques or transparent bridges are used.

We will continue our investigations by setting up two NBA systems, one provided with the original flow data and the other one receiving manipulated flow streams during the learning phase. Thereon, a deeper evaluation of the effects and possibilities when learning in a manipulated environment will be done. The results already found as well as the outcomes of the further attack investigations will flow into our ongoing research for building a Fast Learning ANN based IDS which is able to overcome the hazards of the learning phase. Our approach is a reduction and hardening of the learning phase and the usage of pre-processed neural network elements. To model the hereby needed components, we will use the knowledge gained by the investigation of the possibilities to influence the flow data. Addressing the threat which is given by the manipulation of flow packets or the injection of traffic, the structure shown in Figure 34 is used to build the NBA system.

The initial phase is designed as followed: While in the setup process of the system, an initial scan of the network environment is done using nmap [161]. The initial composition of the neural network will be based upon the selection of modules for the identified services and protocols, taken from a pre-processed database. These pre-processed modules are designed or pre-trained in a secured environment in advance. The administrator setting up the IDS has to confirm the selection of modules therefore no unauthorized service is included falsely in the construction. In the beginning, the operational network is a copy of the initial training network and is not changed in real-time. Yet, there is no consideration of traffic changes due to time of day, etc. and no dependencies among the streams are representable. Therefore, the building of the behavioral model and the compositions between the pre-processed neural network elements have to be learned. To secure this process, a set of policies depending on the detected services and devices in the network is generated and has to be verified by the administrator. For reasons of the security of the learning process, the policies are set restrictive. If there is no anomaly during the learning phase, the training network is transformed to the operational network at discrete times and the learning state is secured. If a critical situation is detected by the verification module, the system administrator will be informed to decide the further steps, e.g. a roll-back of the training network or the continuation of the learning process. Note that the traffic anomaly detection capability of the pre-processed modules is not changed during the learning process, but only the mutual dependencies and timely constrains.

5 Management of large MANETs (MaMANET)

Mobile ad-hoc networks will play a vital role in future mobile networks. Whenever no infrastructure can be used, either because it was destroyed (like in crises operations), there is none present (like in desert areas) or because it should not be used (like in military operations), mobile ad hoc networks can still provide communication means. This technique, however, raises new challenges with respect to the management of such a system. Current management approaches address only some of the aspects identified in that field. Thus, this section addresses the following topics that have been the areas of interest in the MaMANET activity:

1. requirements analysis
2. define a comprehensive management architecture for MANETs
3. detailed part 1: propose service search
4. detailed part 2: resource management of disruption tolerant networks

5.1 Towards a Management System for SDR-based Ad-hoc Networks in Military Environments

5.1.1 Introduction

Software Defined Radio (SDR) offers new possibilities to radio communication. While already available in commercial applications, its use in military deployment is just starting. SDR devices will be able to provide high data rate communication and networking capabilities on the battle-field. They, thus, will constitute a mobile ad hoc network (MANET), i.e. a wireless network without fixed infrastructure, which can be deployed spontaneously over a geographically limited area [162]. The SDR devices may make use of different types of waveforms each exhibiting different physical properties.

Such networks are characterized by extremely high node mobility with consequently dynamically changing network topology and scarce as well as fluctuating bandwidth. Strict requirements on security, reliability and availability combined with the dynamic nature of the network provide a strong motivation for self-forming, self-configuring, and self-healing capabilities of the network [162].

Such networks impose quite a lot of challenges - not only to the network and protocol design - but also to the management. We identify these new challenges, and analyze whether existing management approaches can be used for SDR management. Later on a system architecture for SDR management is outlined.

5.1.2 Scenario

Figure 35 presents an operational military scenario for wireless mobile ad-hoc networks in highly dynamic tactical environments. The scenario consists of five independent networks each operating on a different physical channel by using different waveforms or the same waveform with different channels. An SDR device with more than one line (e.g., two-line or three-line devices) allows connecting the networks. We assume that all communication in the networks is IP-based.

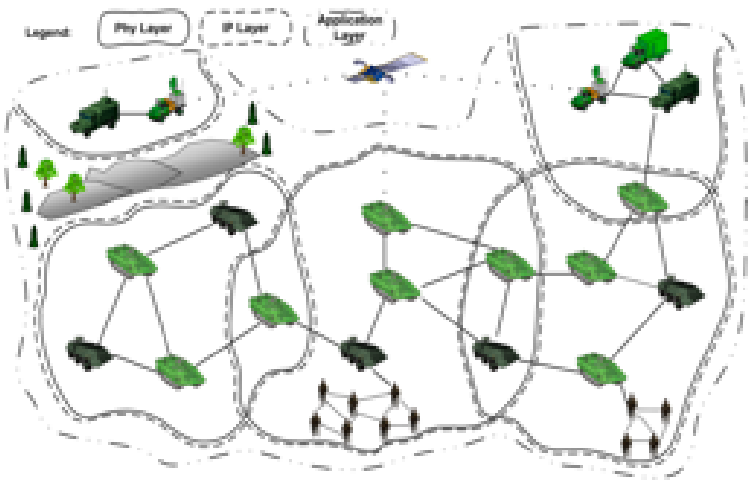


Figure 35: Sample scenario

Each physical network has its own IP address space. Re-addressing mechanisms take care for assignment of appropriate IP addresses when one device roams between the networks. Thus, on application layer only one big network exists.

SDR nodes play multiple roles in the network. On one side they perform all routing tasks within the network, either as a hop in the MANET or as router for an attached LAN (e.g., within a vehicle). On the other side an SDR device is also sender and receiver (source and destination) for all services that run on the device itself. As an assumption all SDR devices are able to encrypt data before sending it over the air. Hereby they create a “black” overlay network. In military terms data is referred to as black if it requires no further protection. The opposite is red data which is classified data and has to be protected against eavesdropping.

5.1.3 Related Work

The management of mobile ad-hoc networks is already addressed by other authors. They, however, have either a different scenario in mind or focus only on selected aspects of management.

The International Standards Organization (ISO) defined as part of their Telecommunication Network Management (TNM) the FCAPS model [163]. FCAPS is an acronym for fault, configuration, accounting, performance and security management and describes the five functional areas a network management system for commercial fixed networks must service. Private mobile ad hoc networks have not been in the focus of the ISO.

ANMP (Ad hoc Network Management Protocol) [164] is an SNMPv3 [165] compatible management protocol. It focuses on efficient and secure transport of management messages. Algorithms for clustering nodes in order to combine and aggregate messages are described. The paper addresses, however, only the protocol and not the management architecture.

The Guerilla Management Architecture proposed in [166] focuses largely on the resource discovery and monitoring aspects of ad-hoc network management. Available network resources and operating conditions such as battery power usage, processing load, and node isolation probability, as well as certain management objectives are modeled as functions and facilitated for management decisions [167].

A management framework to assess the operational state and the behavior of nodes in mobile ad-hoc networks is proposed in [162], however, security and system management aspects are not addressed.

Policy-based management approaches are proposed for ad-hoc networks as well in [167], [168], [169] and [170]. These papers focus mainly on how to use policies for management, less on what policies are necessary.

DRAMA (Dynamic Re-Addressing and Management for the Army) [171], [172] is a policy-based mobile ad-hoc net-work management system. It focuses on the reduction of management traffic overhead by controlling the content and frequency of management information reported, by aggregating and filtering it at the source.

To summarize, none of the previously mentioned systems or other related work address the special requirements of an SDR-based ad-hoc network in a highly dynamic tactical environment. Specific aspects like radio silence, and the separation between black and red management is not taken into account. Furthermore, also other aspects of network, system or security management are not addressed sufficiently.

5.1.4 Functional Requirements

The following section identifies the functional requirements of a management system for mobile SDR-based ad-hoc networks. To undermine some specifics, we structured the requirements into the three areas, network management, system management and security management, and the common areas, planning, visualization, archiving and performance management.

Network Management An SDR device in a mobile ad-hoc network is acting as an intermediate device. Depending on its detailed instrumentation it could be a repeater, bridge, router, firewall, or VPN gateway. Consequently, the forwarding capabilities on layer 1 to 3 of the OSI reference model need to be managed.

On the lowest layer the line or lines of a radio need to be configured and monitored. The most important aspect is the choice of the waveform and its parameters (e.g., frequencies, hop patterns or transmitting power).

Roaming restrictions identify those networks that a radio is allowed attaching to as well as prohibited networks. The management system (MS) needs to ensure, that these constraints are consistent with the configuration of the waveforms.

A management system must be able to create and delete VLANs [173] on an SDR device as well as to add and remove interfaces from a VLAN.

In a mobile ad-hoc network the establishment and tear-down of links is highly dynamic. Tracking all existing links by the management system in real-time will waste a lot of bandwidth and provides only little gain, as the information becomes outdated very quickly. However, even in mobile ad-hoc networks some parts of the topology remain quite constant over a specific time frame. These are, e.g., links between networks (wired and wireless) or links to satellites. Monitoring of these links must performed by a management system.

The management of IP addresses and IP address spaces is also a key requirement. Beside static assignment of addresses, dynamic re-addressing policies may need to be deployed on devices. For a management system it is also necessary to configure routing protocols and static routing entries. If SDR devices support policy-based rout-ing, the MS must support creating and editing of these policies, and ensure their consistency with other configuration settings. All these IP related tasks are necessary for the red network, as well as for the encrypted black overlay network.

A unique requirement for military networks is the enforcement of periods of radio silence. An MS must support issuing of commands for radio silence and inform other nodes about these events. If a period of radio silence is foreseeable, routes should already be adjusted and services relocated if necessary. An important requirement here is not to interpret silence (e.g., no response from a managed device) as a fault, which is the case in traditional network management.

An MS must also be capable of controlling the QoS (Quality of Service) mechanisms of SDR devices. Especially in a dynamic network, where the bandwidth of links changes continuously and a network may break in partitions, a human operator must be supported by automated mechanisms. Even further, thinking of autonomic management, self-configuration capabilities of SDR devices should be encountered to support such controlling of the QoS.

During the introduction phase of SDR, in many deployments SDR devices must coexist with legacy radios. Because of their improved capabilities, SDR devices will most likely provide gateway functionalities between radio circuits of legacy devices and IP-based networks. Mere routing functionalities are not sufficient in this case as many legacy systems are not IP-based and applications use proprietary addressing schemes. The MS must provide the possibility to configure the gateway functionalities and monitor their operation.

System Management As already mentioned above, SDR devices play multiple roles in a network. Their role as intermediate systems has been addressed in the previous section. This section focuses on the role as an end host.

First of all, a MS must support an inventory and asset management, i.e. store a list of all deployed SDR devices with their most relevant properties, e.g., type of device, device ID, installed radio modules and installed software. An administrator should be able to maintain this information manually, but the MS is expected to support this task by automatically supplementing the master data.

In addition to the static master data the MS should automatically collect operational data from the SDR de-vices on a regular basis. This data includes for example the powering of the device (i.e. battery or external powered), the battery level if applicable and a health status of the device.

Looking at the scenario, it is obvious that returning an SDR device to a repair shop is not always possible or at least very demanding. Therefore, as much maintenance as possible must be performed remotely, which has to be supported by the MS. Necessary functions are the distribution and installation of new software and updates, as well as starting and stopping of services.

The MS must support performing these actions batched, i.e. putting the actions in a queue and perform them when a connection and sufficient bandwidth is available, or offline, i.e. by copying the data to a mobile storage, which is later on attached to a radio.

Security Management The third specialized area of the management system is security management with the following sub-tasks: (i) authentication and authorization, (ii) key and certificate management, (iii) VPN management, and (iv) classification management.

On the one hand SDR devices in military scenarios must feature an elaborate authentication and authorization concept consisting of users, passwords, roles, groups and access control lists (ACLs) or capability lists (CAs) as they are processing classified data. On the other hand modify-ing users and roles must be possible quickly if situation requires (e.g., if mission changes or after hostile actions). Authentication and authorization data of one device must be replicated locally, because a connection to the user management (like X.500, LDAP or a database server) or an AAA server (like RADIUS or DIAMETER) might not always exist. The MS must control and perform this replication.

Confidential radio communication must be encrypted. This requires appropriate keys or certificates, respectively, to be available at the radios. Those keys and corresponding revocation lists must be distributed by the MS. If appropriate interfaces to already existing key management systems must be implemented.

Furthermore, policies need to be created and distributed, governing which key and encryption scheme to apply to what data. In other words these policies describe which VPNs to establish or join.

A special requirement for military communication equipment is the handling of different classification levels, which is also termed red-red-separation: Device can operate with data of different classification levels (e.g. secret or confidential) classified by different organizations (e.g. national confidential vs. NATO confidential) at the same time. Policies need to be specified defining which data is allowed to be forwarded between connected networks.

To address intrusion detection, SDR devices and the management system need to implement policies and situation management (i.e., awareness), which is also the basis for anomaly detection. The MS must support distributing new policies and situation information to the SDR devices and revoking old information.

The MS must store all these configuration changes until the respective device is reachable and the action can be performed. It must log if and when a device was updated, and should display to the administrator a snapshot of the differences between the target and actual state.

But not only one device might operate with data of different classification levels, also the MS itself faces such a challenge. On the one hand it must be able to process highly sensitive management information, e.g., hop pat-terns or frequencies, and configure classification relevant parameters, e.g., which encryption algorithm and key to use, while most management data is at a much lower classification level. As a compromise between security and usability we propose to process the highly sensitive information on dedicated, specially secured, systems only and choose a lower classification level for the management system as a whole. Distribution mechanism of the MS might still be used if classified information transferred is encrypted.

Visualization The visualization module needs to provide different views of the system. A topology view, well known from management systems for fixed networks, is only one of these views. As stated above a detailed topology view is only of limited use in mobile networks.

Displaying the utilization of ad-hoc links in real-time or even in near real-time is hardly possible, too. The bandwidth of a link can change abruptly, e.g., when mov-ing and an obstacle is left behind or a new member joins the network and takes its portion of the shared medium air.

But still a graphical overview of the configured networks, its gateways and the uplinks to fixed networks, including satellite uplinks, has to be provided. Thus, autotopology and autodiscovery need to be supported. Utilization or available bandwidth on mobile links should be shown on a per network level. This enables performance management on a mid-term time scale (see also Section 4.7).

A management system needs to visualize all alerts and events reported by the SDR devices. This must be a common function, i.e. the status of the system as a whole must be composed of the status of all three specialized areas described in the previous sections. Yet, filter mechanisms hiding events of various types or specialized maps contain-ing only a subset of events are required to help the operator to get a better picture of the relevant facts. Event correlation in such a dynamic environment requires also new approaches.

Usually, military SDR devices are equipped with at least one GPS receiver. A nice feature for the network administrator is the display of all radios on a real map. How-ever, the functional usefulness of such a view is question-able and because of its operational sensitivity additional protection is necessary.

Planning Beside the visualization of the current status of the system, a module is necessary for planning structures and future configurations.

The MS needs to support the planning by simulations and plausibility checks. An operator must be able to store planned scenarios, copy them and modify them later on. In order to put plans into practice quickly and efficiently, an MS must support creating operational configuration settings out of these scenarios and distribute the settings directly or batched as described above.

Logging and Archiving All events from SDR devices and configuration changes must be logged by the MS. This includes also log files from devices and services, as far as they are transferred to the MS. This data should be used by online and offline Intrusion

Detection Systems (IDS) in order to detect security bleaches. Such data is also a valuable source for forensic analyses.

Performance Management Performance management is responsible for assessing and improving the effectiveness of the network and its end systems. Performance management has to include all planning intervals, from short-term over mid-term to long-term.

Within short-term management exceptional burden on single hotspots could be eased by configuration changes, e.g., routing changes or relocation of services. In a mobile ad-hoc network this is only possible to some extent since mobile links and the network topology changes too fast. Only for network gateways or satellite uplinks short-term performance management is possible.

Mid-term management is closely related to planning. An administrator has to plan networks, number of nodes in a network, waveforms, frequencies or service locations under the given restrictions of a mission in the most efficient way. Data in the archive contains the experience from past deployments and may be used to support the planning process.

Long-term performance management refers to capacity planning [174]. It addresses the task of determining the likely future resource requirements. Archived data could be used to extrapolate future traffic profiles and to identify bottle-necks in the network where improvements are of highest priority.

5.1.5 Design Requirements

Beside the functional requirements, as described in the previous chapter, some general design requirements need to be met by a management system for SDR-based net-works. These requirements apply to all areas described above.

The management system gains exceptional importance because communication on today?s battlefields is of utmost importance, and a loss or even a disturbance of communication can have serious impacts. This requires sophisticated security mechanisms within the MS:

- Access to the MS must to be protected. Each user working with the MS must be authenticated, and all actions performed must be authorized by the management system and by the managed object.
- Encryption of messages within the management system is desirable, but not a requirement, as all wireless links within the network are already encrypted. All other links are protected against eavesdropping by construction.
- Message integrity must be ensured end-to-end in order to detect manipulation of messages.
- Non-repudiation mechanisms must be in place in order to chain the manager and his configuration actions together.

The limited bandwidth in wireless networks (especially in military ones) requires efficient communication (i.e., minimal overhead). This should be taken into account when choosing a management protocol and architecture.

Network management may be performed on different military levels (e.g., battalion, brigade, or division) with different tasks and with different levels of detail: while the brigade staff manages satellite uplinks and connections to and between its battalions, the battalion staff manages its networks, frequencies and the routing within its networks. In this scenario the higher levels only need aggregated in-formation from lower levels. The number of levels, the tasks on each level, and the aggregation detail must be freely configurable, as the spectrum of military missions is very diverse. For a temporary task force operation it is possible that, e.g., no management staff is in-theater at all, and all management activities are performed by the operations command in the home country via a satellite link.

Although upper layer MS in a management hierarchy regularly get only aggregated information from lower layers, it must always be possible to retrieve detailed data on request. E.g., in special operations mission with a high potential of political implications the detailed status of the communication system might be of interest even for the highest command.

As shown in Section 2, mobile networks have connections to other wireless and fixed networks. SDR devices in to role of routers will connect LANs in command posts or vehicles directly or indirectly to these networks. For an efficient operation of the entire network Interfaces to and interoperability with management systems of adjacent networks are essential.

5.1.6 A Management Architecture for SDR

Based on the identified requirements, the following architecture is derived. The system is distributed, consisting of management stations, proxy stations in the network, and the SDR devices, which will be managed.

The architecture is shown in Figure 36. Data Collection and Distribution Repositories (DCDR) represent the lowest level of the MS. They retrieve status information from the devices, collect asynchronous event messages, apply configuration changes on the devices and distribute soft-ware. At one time exactly one DCDR is responsible for an SDR device. This assignment, however, might change dynamically reflecting changes in network topology or mask-ing temporary or permanent failure of a DCDR.

The Data Convergence Layer (DCL) implements location transparency and caching for higher layers. The DCL is an abstract component consisting of its different in-stances on each management station. If higher layers (i.e. management applications via the access control layer) re-quest data from SDR devices the DCL first checks if this data is not already present in its cache or the cache of any DCDR. If it can be found there, the data is accessed from cache, and thus no valuable bandwidth of the SDR net-work is wasted. If it is not present, the DCDR in charge of the SDR device retrieves the data from the device.

Furthermore, the DCL distributes updated configuration and software for one device to the assigned DCDR, which then deploys the updates on the device or stores them for later deployment if the device is not reachable at the moment. Important updates might

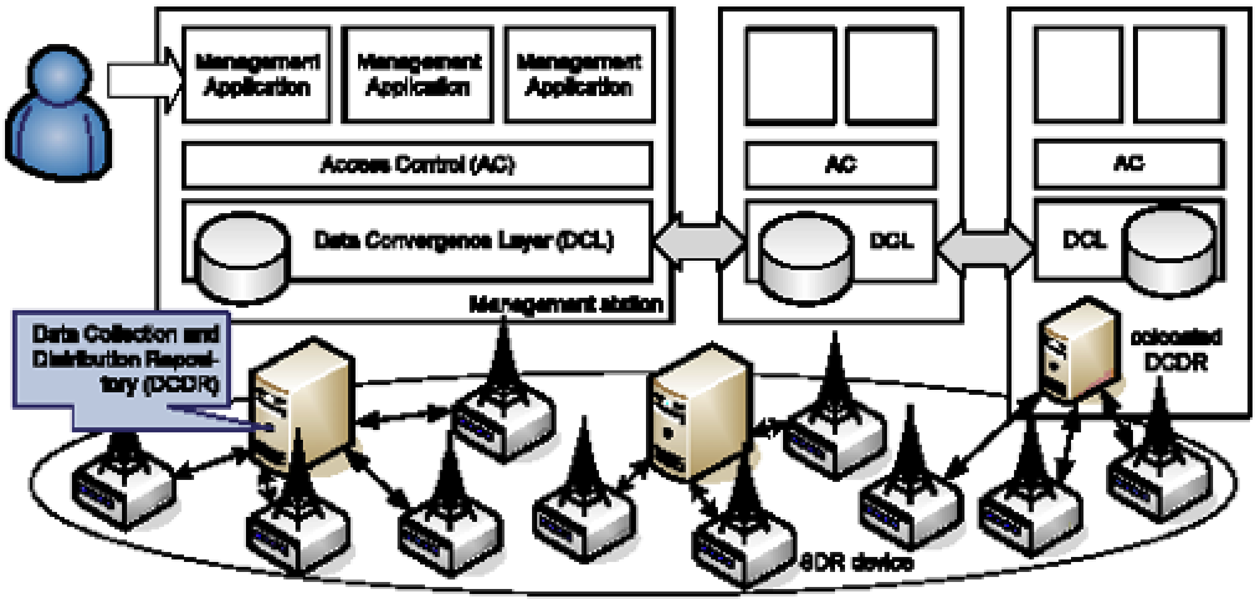


Figure 36: Management system architecture

be distributed to multiple DCDR in order to increase the probability that the device will sometime have access to the information.

On each management station a platform framework is deployed, consisting of a local instance of the DCL, an Access Control module and an application framework. The Access Control (AC) module resides above the DCL and checks for sufficient permissions of performed actions. Management applications implement the identified functions of the MS and present a graphical user interface to the operator.

Data Convergence Layer (DCL) The Data Convergence Layer connects management stations to each other and to Data Collection and Distribution Repositories (DCDR). Management applications do not need to be aware of the distributed nature of the management system. They only access the local instance of the DCL, which finds and gets the requested data or deposits data to be distributed in the respective repositories. This allows each management station to access all data and all devices (if permitted). Even normally a set of SDR devices is managed by only one management station there will be situations where this fixed mapping does not hold (e.g., in case of failure of one management station or in special operations).

The DCL is organized decentralized without any central infrastructure and therefore without a single point of failure. Distributed hash tables (DHT) are used to index and query data and to allow joining and leaving of management stations and DCDRs. DCL instances monitor their neighbors and DCDRs in order to detect failure.

The DCL also monitors the assignment of SDR de-vices to DCDRs and if necessary restructures it. A tradeoff between overhead cause by frequent updates and inefficient DCDR assignment is necessary.

Data Collection and Distribution Repository (DCDR) DCDRs might be collocated with management stations but are logically independent components. They might also be installed on other servers or on SDR devices. DCDRs optimize the communication between device and management station. In a first step management stations are only interested in the overall status of the systems. If everything is 'green' only less additional information is necessary. But if the status degenerates more information for isolating the source of problems is necessary.

This optimization is provided by the DCDRs. They summarize the information retrieved from the devices and transfer only aggregated data to the management stations. In case the management station requests more detailed data, the DCDR will serve this request from its cache.

Assessment The architecture fulfills the design requirements put forward in Section 5. Even the architecture is not a hierarchical one the flexibility of the DCL allows a hierarchical configuration. On management stations of lower layers of the hierarchy management applications need to be in-stalled, which aggregate data for stations on a higher layer and distribute them with the help of the DCL.

Classified data will only be transferred encrypted from the devices via DCDR and the DCL to special management applications which are able to decrypt the data. These management applications are installed on specially protected stations, which may even be different networks.

DCDRs optimize the communication by pre-aggregating information close to the SDR devices and by helping distribute software and configuration changes.

Interfaces to other management systems are implemented as management applications. Thus several different interfaces may exist at the same time and further inter-faces may be implemented without changing the architecture.

5.1.7 Summary and Conclusions

We analysed the requirements towards a management system for mobile ad-hoc networks consisting of SDR devices in a military deployment. The requirements are divided in three functional areas which are closely re-lated to each other and design requirements which apply to the whole system. Afterwards, a highly flexible and distributed architecture for SDR management is proposed. Peer-to-Peer distributed hash tables are used to improve the robustness of the system. Such new approaches are necessary because traditional centralized and hierarchical concepts do not cope with the necessary dynamics and complexity. First evaluations in the test bed have already been performed. Future work will address further issues like the differences and synergies, and possible interaction be-tween autonomic management and autonomic networking, or how much management functionality should be built into the network, and how much should be implemented outside. A further step is incorporating the MS in a Security Management Infrastructure (SMI). An SMI, also known as Enterprise Security Management system (ESM) [175], provides a holistic view on security, which is an absolute must as otherwise only specific aspects of security, leading to a locally limited optimum, may be addressed. Further-more, SMI comprises the systems and resources required to order, create, disseminate, modify, suspend, and terminate management controls providing and operating security services, processes, and devices across the enterprise according to a security guideline [176]. This is necessary as current network environments have an ever-increasing variety of security technologies, mechanisms and security objects in order to meet security requirements [177], [178].

5.2 Service Search

In informal data sharing environments, misspellings cause problems for data indexing and retrieval. This is even more pronounced in mobile environments, in which devices with limited input devices are used. In a mobile environment, similarity search algorithms for finding misspelled data need to account for limited CPU and bandwidth. Similarity search for service discovery can significantly improve service management in a distributed environment. As services are often described informally in text form, keyword similarity search can find the required services or data items more reliably. The prototype shows P2P fast similarity search (P2PFastSS) running on mobile phones and laptops that is tailored to uncertain data entry and uses available resources efficiently and P2PFastSS is suitable for similarity search in large-scale network infrastructures, such as service description matching in service discovery or searching for similar terms in P2P storage networks.

5.2.1 Activity

This prototype has been implemented and shown with real devices at the CCNC 2009 conference in the demo session [179] and updated and refined since then. Thus, this text is based on the paper submitted to that conference.

5.2.2 Introduction

Few of the known distributed hash table (DHT)-based search methods support similarity search on keywords. Approximate keyword search is essential, as misspellings and spelling variants make the localization of required information a difficult problem. Without spelling correction, approximately 10% of all queries are not found, because of typos or misspellings [180]. Mobile devices usually have a limited keyboard for textual input, which makes misspellings more likely. Machine learning methods, as applied by Google are not always applicable, as they require a large corpus of queries.

In DHTs, the main operations are *put(key, value)* and *get(key)*. Those operations, in most cases, require $O(\log n)$ messages to be sent in a network with n nodes. Similarity search algorithms for structured P2P networks have been proposed by Ahmed et al. [181], introducing a routing algorithm based on Bloom filters and Wong et al. [182], introducing a routing algorithm in a keyword metric space. However, P2PFastSS [183] is the only algorithm that supports fast similarity searches that runs on existing DHTs without modifying the routing algorithm.

In a real world scenario, a download service on a mobile device, which serves files to other, is offered. Content of this service includes text files or media files with meta information. Other users that are on the same network can use the similarity search to find files. Similarity search is necessary because of misspelled content and user queries using the download service.

5.2.3 Mobile P2P Fast Similarity Search and Demonstration Scenario

For P2PFastSS, the same concept of deletion neighborhood is applied and the neighbors from the target are stored in a DHT. Using P2PFastSS for the example *test* and *fest* would result in the storage of *fest, est, fst, fet, and fes* using the put operation of a DHT.

The following example shows the indexing of the keyword *test* pointing to the document with DocID: 0x321. Keys in *get* and *put* operations are usually generated using a hash function $key = h(string)$. Node 0x1 first generates the neighbors of *test* with $k = 1$ (*test, est, tst, tet, tes*). All neighbors are indexed in the DHT. Figure 37 shows the indexing of *est*. First, node 0x1 looks for peers with an id close to the id of the neighbor where *est* should be placed, *h(est) = 0x123*. Node 0x4 replies in step 1 with the address of node 0x24, which is closer to 0x123. In step 2, node 0x24 replies with the addresses of nodes 0x124 and 0x122, which are closer to id 0x123. In step 3, these nodes will store the keyword and the document reference for *test*. The keyword is stored redundantly to provide robustness even in case of node failure.

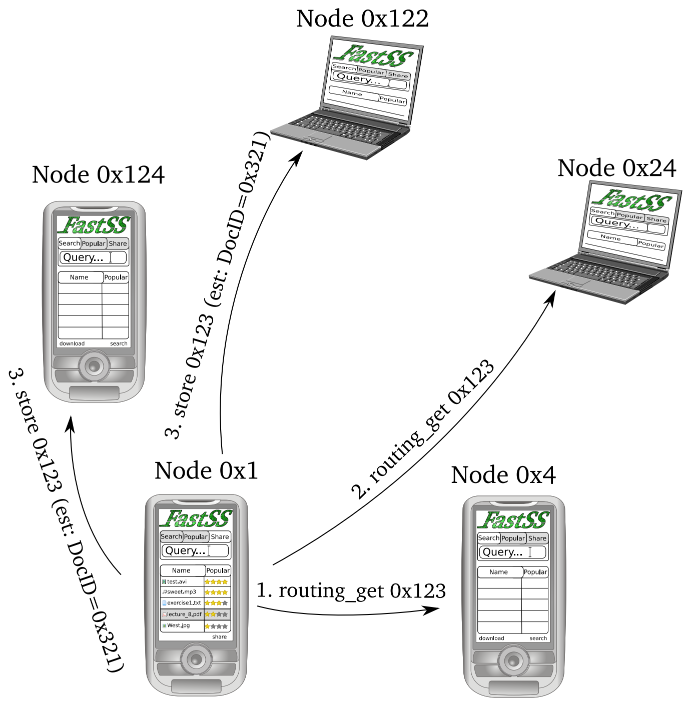


Figure 37: Indexing of *est* with id=0x123

Figure 38 shows a query execution. Node 0x1 queries for the keyword *fest*. First, neighbors are generated (*fest, est, fst, fet, fes*). In steps 1 and 2, close nodes are queried for neighbor *est*, with id 0x123. In step 3, neighbor 0x124 which stores an index entry for neighbor 0x123 replies with the reference to document 0x321. This document contains the keyword *test*, and as $ed(test, fest) = 1$ the document or a preview of this document is shown to the user.

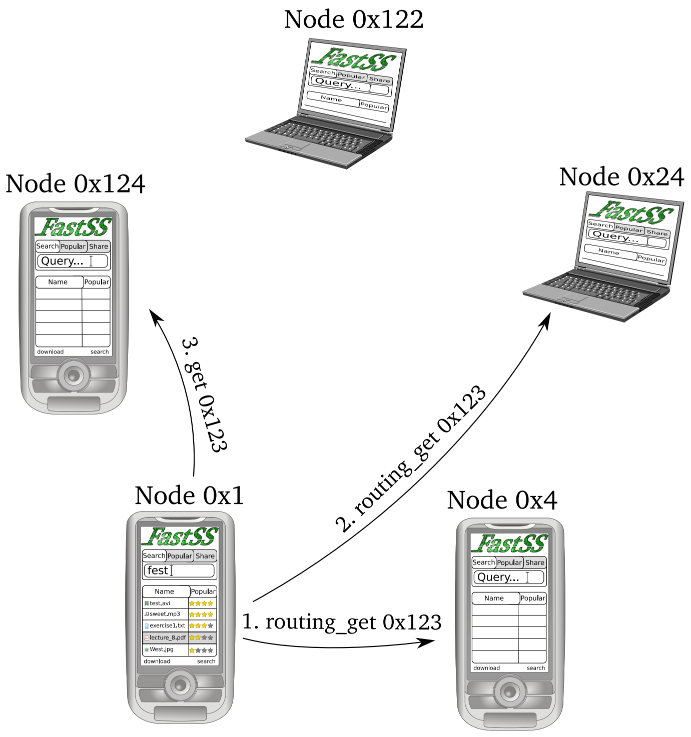


Figure 38: Querying for *est* with id=0x123

The demonstrated prototype of mobile P2PFastSs implements the following three layers. The top layer is the user interface for document or content search. The undelying P2PFastSS layer offers two operations, *index* and *search* and carries out neighborhood generation, indexing and searching. The underlying DHT layer offers *get* and *put* operations. The DHT layer operates on top of Google's Android, which can store data persistently, play media files, and use WiFi.

In this prototype, five users equipped with handsets share documents in a local network. As content, Wikipedia articles and multimedia files are used. Articles are written and indexed by users. From the content, which is published on the local network, keywords are extracted and neighbors are generated. Each neighbor holds references to other devices, subject to a timeout. A user searches for content using keyword search. A keyword search may contain misspellings up to $k = 1$.

A search for a keyword in mobile P2PFastSS returns IPs of mobile devices that host files with a similar keyword. Using P2PFastSS to search for services, the indexed document is the service description. In such a service search scenario, a user that searches for a service retrieves a list of IPs that contain a similar keyword in the service description to the search query. Thus, the presented scenario shows the practicability of mobile P2PFastSS for service discovery and search.

5.3 A Multi-Radio Power Management Scheme for Disruption Tolerant Networks

5.3.1 Introduction

Disruption tolerant networks (DTNs) is a research area aiming at developing network communication when connectivity is intermittent and prone to disruptions. In general, DTNs are applicable in remote and hazardous areas where the energy sources are constrained. They are also assumed to operate over a long period of time. Therefore, several research efforts have been started to develop power management schemes for disruption tolerant networks.

In DTNs, nodes need to discover neighbors to establish communication. Searching for neighbors in sparse DTNs can consume a large amount of power compared to the power consumed by infrequent data transfers. Therefore, novel power management schemes are needed to address this problem and to save energy in the neighbor searching mode. Designing such power management schemes is challenging because nodes need to know when to sleep to save power and when to wake up to search for neighbors. Ideally, power management schemes should not reduce network connectivity opportunities, which would negatively affect the overall performance of the network.

One such class of power management schemes for minimizing the use of idle listening mode are the wake-up based schemes in multi-hop wireless sensor networks, which can basically be categorized as rendezvous, asynchronous or on-demand [184]. The rendezvous schemes require all sleeping nodes to wake up at the same time. However, these schemes are not easy to implement because synchronization between nodes may be difficult to achieve and energy may also be spent on synchronizing the nodes. On the other hand, asynchronous schemes do not require time synchronization between the nodes and are designed to operate by having overlapped active periods for neighboring nodes within a specified number of sleep cycles.

On demand schemes make use of multiple radios; a low-power radio for discovering nodes in the neighborhood and a high-power radio for carrying out data delivery. Until recently such on-demand schemes have not been viewed as being useful for DTNs since low-power radios did not have long transmission ranges, which resulted in poor network connectivity. However, with the advent of low-power radios that offer a similar transmission range to those of the high-power radios, on-demand schemes for DTNs are now worthy of further investigation.

A new power management scheme is developed in this work. This scheme tries to save energy in searching mode with a minimum degradation of the overall network performance. It is based on the context aware power management (CAPM) scheme [184], and it also uses an additional low-power radio interface (LPR) [185]. The rest of this section is structured as follows: Related work is described in Section 5.3.2. Section 5.3.3 presents an overview on the PROPHET routing protocol that is used in our MR-CAPM scheme. It also shows an overview about the energy consumption. The MR-CAPM scheme is described in Section 5.3.4. Section 5.3.5 discusses the simulation setup. The MR-CAPM performance evaluation is shown in Section 5.3.6 before the work concludes in Section 5.3.7.

5.3.2 Related Work

Several power management schemes have been proposed for ad hoc networks. These schemes allow nodes to disable their radios when they are not used to save energy and to prolong the network life time while keeping network connectivity [186, 187]. Unlike our work, these schemes assume that a node has another node within its communication range most of time.

There have been several efforts to develop energy efficient medium access control (MAC) protocols [188, 189]. These efforts are motivated by the observation that transmissions of multiple nodes in a wireless network may interfere with each other. To avoid this interference, only two nodes can communicate with each other at any point in time. Therefore, significant energy can be saved if a node sleeps while others communicate, thereby in many cases improving overall network performance. These MAC protocols propose mechanisms to increase sleeping time based on the traffic in the neighborhood. However, these MAC protocols are designed for dense networks rather than sparse networks.

Recently, approaches for power management of sparse disruption tolerant networks have been developed. These approaches focus on saving energy in searching mode when nodes search about each other to communicate. Jun et al. [190] presented a power management scheme that assumes synchronized clocks and allows nodes to be in one of three modes, dormant (sleep) mode, search mode, and contact mode based on knowledge of future contacts. There are three levels of knowledge:

- **Complete Knowledge**: nodes know everything about their future contacts so they know exactly when to wake up and when to sleep.
- **Zero Knowledge**: nodes have no information about each other and so they need to search about each others.
- **Partial Knowledge**: nodes wake up and sleep based on probability metrics derived from statistical information.

In a subsequent paper, Jun et al. [185] proposed to minimize the power consumed in searching mode by using an additional low-power radio to discover contacts and to awake the high-power radio to undertake the data transmission. However, both works have the same disadvantage since nodes need to be equipped with a Global Positioning System (GPS) receiver to achieve synchronization, which increases the network cost. GPS also only works on surface outdoor and it can not be used for indoor applications.

Banerjee et al. [191] present a hardware and software architecture for energy efficient throwboxes in DTNs. These throwboxes are stationary battery powered nodes with a radio interface and a storage device platform. This scheme has many disadvantages such as: the expensive cost of the network since each node needs to be equipped with a Global Positioning System (GPS) receiver to achieve synchronization. It is efficient only for the DTN scenarios that allow using stationary nodes as well for predictable and semi-predictable movement patterns such as vehicular networks movement.

The Context Aware power Management Scheme (CAPM) has been developed by Chuah et al. [184]. It is an asynchronous mechanism in which each node works on its own wake-up schedule independently. The CAPM scheme has a fixed duty cycle which consists of a wake-up and a sleep period. Each node wakes up for a fixed or adaptive period and sleeps for the remaining time. However, this is the only work that shows a comparison to the other existing power management schemes. The authors compared the energy efficiency of the CAPM scheme to the PSM scheme described in the Hierarchical power management scheme [185]. The comparison results indicate that CAPM can achieve a 80% energy saving and yet achieve a delivery ratio and an average delay that are comparable to the case without power management (no sleeping), While the PSM scheme can only achieve a 40% energy saving.

Unlike the other DTN power management schemes, Sadler et al. [192] present an approach that derives energy savings by implementing lossless adaptations of compression algorithms on the source nodes generating data for delivery. These algorithms save energy by decreasing the data volume that needs to be transmitted over the wireless interface. They use an initial dictionary with 256 entries for all 8-bit characters. Since both the sender and receiver have the initial dictionary, new entries are created based on existing entries without any additional transmission overhead. However, each node is equipped with a GPS receiver to achieve synchronization and this is considered a disadvantage since it increases the cost of the network.

5.3.3 Background

This section presents an overview on the PROPHET, which is the routing protocol that is used in this work. It also shows an overview about energy consumption.

Routing Protocol

Routing in DTNs is a topic of high importance due to the frequent partitions and intermittent connections that are characteristic of these types of networks [193]. In recent years, several routing protocols that address specific issues in DTNs have been proposed [194, 195]. The PROPHET routing protocol [196] was chosen to be implemented for the evaluation of our new MR-CAPM scheme since it maximizes the connection opportunities in mobile networks that suffer from connectivity problems.

PROPHET is a probabilistic routing protocol which uses a history of encounters and transitivity in order to determine future contact probabilities. PROPHET is used for intermittently connected networks where there is no guarantee that a fully connected path between source and destination exists at any time.

The PROPHET algorithm relies on calculation of delivery predictability in order to forward messages to the most reliable node of which current knowledge exists. The calculated probability is used to decide if a certain node is reliable to forward messages to. This probability is calculated using a history of encounters, a data aging mechanism, and the transitivity property of network connections.

Whenever a node is encounter, the probability metric of node A encountering B is updated. As such, a node that is encountered more often has a higher delivery predictability than others. The probability of A encountering B can be calculated using Equation 1. $P_{encounter}$ is an initialization constant with a value between 0 and 1.

$$P(A,B) = P(A,B)_{old} + (1 - P(A,B)_{old}) \cdot P_{encounter} \tag{10}$$

If a pair of nodes do not encounter each other during a time interval, they are less likely to be good forwarders of messages to each other and thus the delivery predictability values must be reduced. Such an aging of knowledge is very important in order to constantly keep the most likely to be successful routes to be the ones that are selected. The aged delivery predictability values for node A encountering B can be updated according to Equation 2. The parameter γ is the aging constant with values between 0 and 1, and K is the number of time units that have elapsed since the last time the metric was aged.

$$P(A,B) = P(A,B)_{old} * \gamma^K \tag{11}$$

There is also a transitive property in delivery predictability, which is based on the observation that if node A frequently encounters node B and node B frequently encounters node C, then node B probably is a good node to forward messages destined for node C from A. This transitive probability may be calculated using Equation 3. The parameter β in Equation 3 is a scaling constant with values between 0 and 1 that controls the impact of the transitivity property on the delivery predictability.

$$\begin{aligned} P(A,C) &= P(A,C)_{old} + [1 - P(A,C)_{old}] \\ &\quad \cdot P(A,B) \cdot P(B,C) \cdot \beta \end{aligned} \tag{12}$$

When two nodes meet, they exchange the delivery predictability information they currently have stored. This information is used to update the estimated delivery predictability to the destination. A message is transferred to the other node if the delivery predictability of the destination of the message is higher at the other node.

The values for $P_{encounter}$, γ and β are set to 0.75, 0.98 and 0.25 respectively according to the values suggested by the authors of the PROPHET protocol.

Energy Consumption

Three different radios are presented in this work, a high-power radio WaveLan 802.11, and two low-power radios: the XTend and the CC1000 [185, 191, 197]. Table 15 shows the energy consumption of each radio activity (transmitting, receiving, idle, and sleeping) for the three radios. From Table 15, we can see that, a node consumes less energy in sleeping activity than idle activity, while it consumes almost the same energy in both idle and receiving activities. In addition, the XTend and the CC1000 radios consume an order of magnitude less energy than the WaveLan 802.11 radio for each activity. Thus, they can discover contacts using substantially less power. Table 16 shows the bit rate and the outdoor range for the three radios.

Table 15: Energy consumption characteristics of different radio types (in Watt)

Radio Type	Transmit	Receive	Idle	Sleep
WaveLan	1.3272	0.9670	0.8437	0.0664
XTend	1	0.36	0.36	0.01
CC1000	0.0781	0.0222	0.0222	0.00003

The introduction of adaptive radios with multiple transmission modes, at least a high and a low-power mode, has also seen development of power management schemes that leverage this feature by performing neighbor discovery using an asynchronous scheme in the

Table 16: Physical characteristics of different radio types

Radio Type	Bit Rate	Range
WaveLan	2 Mbps	250m
XTend	115.2 Kbps	1000m
CC1000	76.8 Kbps	50m

low-power mode and then switching to a high data rate transfer mode once there is deliverable payload available [198]. Though this method reduces the overall energy consumption, it is worth noting that it also reduces the overall node discovery opportunities since the mode change occurs during the wake-up cycle. This happens because the radio frequency being utilized for node discovery in low-power mode will always be different from the frequency for data delivery in high-power mode and the radio is capable of using only one frequency at a time. This result further strengthens the case for using two radios in DTNs.

Results from initial investigation showed that it is better to use high-power radios for data transmission since these radios normally provide a high data rate. However, since the idle-power consumption for high-power radios is higher than those of low-power low-data-rate radios, they are unsuitable for use during the neighborhood discovery phase. Conversely, the comparatively low energy consumption in idle listening state makes the low-power radios more suitable for neighbor discovery.

5.3.4 The MR-CAPM scheme

The MR-CAPM scheme is an extension of the CAPM scheme [184]. By utilizing two radios instead of one, as in CAPM, we are able to implement an asynchronous on-demand power management scheme that eliminates the idle time of a single high-power radio and only allows the high-power radio to consume power in the sleep mode or while it is transmitting and receiving data. While the high-power radio is only called upon to perform data delivery when necessary, the low-power radio remains active following the original CAPM scheme for neighbor discovery.

Since the neighbor discovery and data delivery radios are now separate, it is extremely important that the low-power radio is chosen with great care. The low-power radio is tasked with neighbor discovery and if the transmission range of this radio is greater than the high-power radio then there might be many delivery failures as nodes discovered by the low-power radio would not always be reachable by the high-power radio. Furthermore, there must be some frequency band independence between the two radios in order to avoid signal interference that may degrade network performance. For testing the MR-CAPM scheme the XTend and CC1000 low-power radios were chosen since their range and power consumption values are such that the high-power radio could still deliver data in every discovery scenario with energy savings. The WaveLan radio is used as the high-power radio since it was also used in the CAPM study and allows for direct comparison with that scheme.

The original CAPM scheme uses the PROPHET [196] routing protocol and to maintain comparability the MR-CAPM scheme is also evaluated using the same routing protocol.

The rest of this section provides details on neighborhood discovery and data delivery phases of the MR-CAPM scheme.

Neighbor Discovery

Neighbor discovery in the MR-CAPM scheme is extremely important in order to decide which nodes can be used as potential next-hop nodes for data forwarding. The nodes discovered in this part of the scheme are added as contacts in the routing table along with a delivery predictability calculated using the PROPHET algorithm. In case of the MR-CAPM scheme, only the low-power radio is utilized to find neighbors.

Each node periodically wakes up for a period W in a fixed duty cycle of length C as shown in Figure 39. Each time a node wakes up, it transmits a beacon containing it's node identifier. In case the node has data available for delivery, it piggybacks a delivery notification to the discovery beacon message. This delivery notification contains information about the destination nodes for the data to be transmitted. In case another suitable next-hop node has its low-power radio active, it replies with the delivery acceptance message to the node sending the delivery notification. After K duty cycles have passed, the nodes let their low-power radio remain active for the full cycle length C, before reverting back to the regular scenario. We refer to the tuple (W,C,K) as the sleep pattern [184].

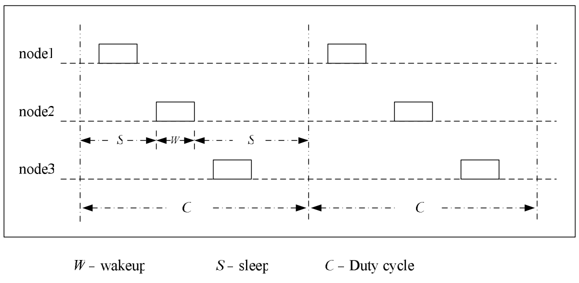


Figure 39: Random wakeup with fixed duty cycle [184].

Figure 40 depicts some of the neighbor discovery scenarios that may be encountered. In Figure 40 (a), node 1 wakes up its low-power radio and transmits a discovery beacon message which is not received by any nodes since they are all inactive (sleeping) at this time. However, the discovery beacon for node 2 is received by node 1 and both nodes 1 and 2 receive node 3's discovery message causing all nodes to be aware of node 3, only node 1 being aware of node 2 and no one aware of the existence of node 1 in their neighborhood. Figure 40 (b) extends this scenario by depicting a case when node 1 piggybacks a delivery notification to its beacon. Since no nodes are active when the beacon is sent, no node sends a data acceptance message. In Figure 40 (c) node 2 broadcasts a data delivery notification along with the beacon and receives an acceptance from node 1 since only that node is active when the beacon is sent. In Figure 40 (d) both nodes 1 and 2 respond to node 3's delivery notification since they both are active.

The neighbor discovery phase is not only used for signaling for data delivery, but also to calculate the delivery predictabilities to nodes in the network to determine the best next-hop using the PROPHET routing protocol.

Data Delivery

Once a delivery predictability has been calculated, the next-hop node identified and a delivery acceptance is received. Then, each node wakes up its high-power radio in order to undertake the data transmission. Figure 41 denotes a scenario when node A and

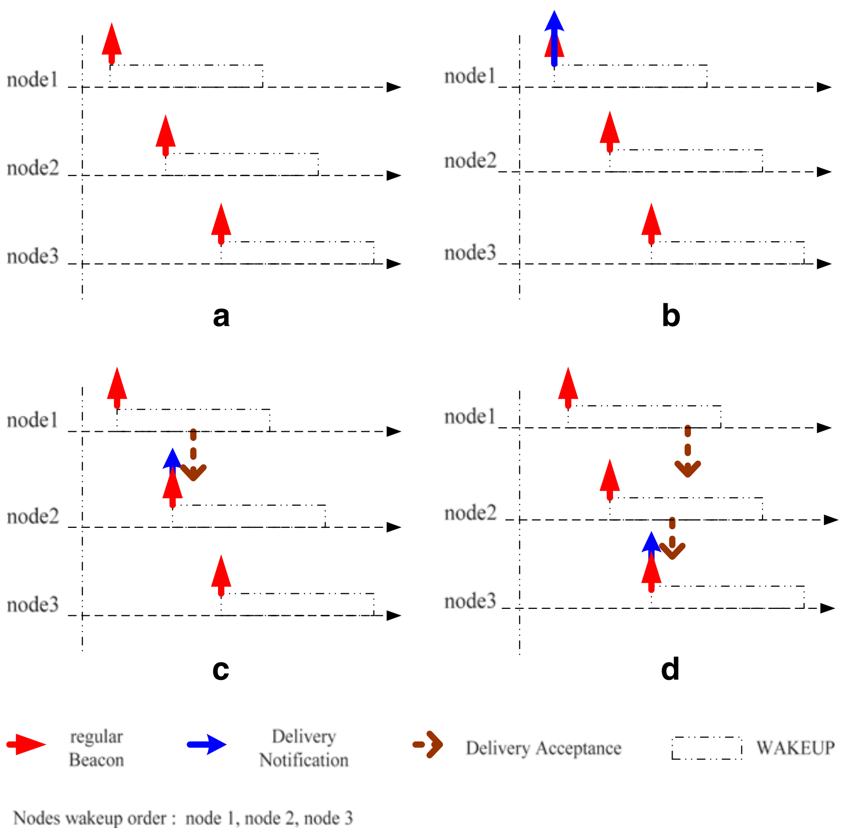


Figure 40: An illustration of the multiple possible neighbor discovery scenarios in the MR-CAPM scheme [184].

node B are both still in their active beacon periods and a data delivery occurs in parallel. However, it is important to note that unlike in the CAPM scheme, MR-CAPM does not require nodes to extend their awake period W in order to be able to receive a message since this will be received by the high power radio. Each node only requires that the delivery acceptance message be received before the end of it's awake period since that is a part of the beacon phase.

Following this scheme ensures that the power usage impact of the high power modem is minimal since it is only active during data transmission and reception phases and at all other times it is in the sleep mode. This does not just reduce the idle wait time but completely eliminates it.

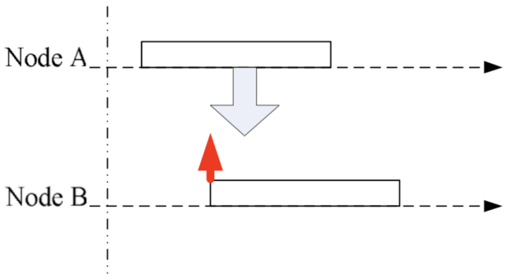


Figure 41: An illustration of the MR-CAPM data delivery scenario.

5.3.5 Simulation Setup and Scenarios

The MR-CAPM was evaluated using ns2 since it is one of the most popular mobile wireless networking simulators available. However, the current versions of ns2 only supports a single radio interface implementation on the mobile nodes and this made it necessary to extend ns2 such that it would support multiple interfaces in order to implement MR-CAPM.

In order to compare our results with the CAPM scheme, we use the same simulation scenario in [184]. The scenario is set up with 40 mobile nodes that are distributed over 1000 x 1000 m^2. Each of these nodes is setup to move at a constant velocity of 5 $\mathrm{m/s}$ following the RWP mobility model. The simulations were also performed with the nodes distributed in an area of 2000 x 2000 m^2 and 3000 x 3000 m^2. As in the CAPM scheme, we use constant bit rate traffic with 10 CBR flows and a packet size of 512 bytes. The traffic generation for each flow varied from 0.25 pkts/s to 3 pkts/s. The sources and destinations of the CBR flows are randomly selected before each run amongst the 40 nodes, however, only a maximum of 10 connections are allowed during each run.

Each simulation runs for 1600 seconds, with 1000 seconds being utilized as a warm-up period and the performance data being recorded only for the last 600 seconds of the simulation. In order to minimize the possibility of only a corner case being encountered due to the RWP mobility model, every reported result is an average taken over 5 runs. The radio models from Tables 15 and 16 are used for all the simulations.

5.3.6 Performance Evaluation

In order to evaluate the performance of the MR-CAPM scheme, we utilize three metrics:

1. Normalized Energy Consumption: The ratio of the energy consumption when MR-CAPM protocol is applied divided by the energy consumption in the absence of power management.
2. Delivery Ratio: The ratio of the successfully received number of data divided by the number of the total delivered data.
3. Average End-to-End Delay: The average delay it takes to deliver a message from the source to the destination.

To obtain a direct comparison with the CAPM scheme we use the same sleep pattern (W,C,K) that is used in the CAPM paper. The rest of this section discusses our experiments.

MR-CAPM scheme evaluation with the optimal values of (W,C,K)

In this experiment we test and evaluate MR-CAPM with the optimal values of the sleep pattern (W,C,K) that are discussed in CAPM paper, more details about these optimal values can be found in [184]. We use 40 nodes distributed over 1000 x 1000 m^2, 10 CBR flows and each flow generates 3 pkts/s. We first evaluate our MR-CAPM scheme using the CC1000 radio as low-power radio and the WaveLan 802.11 as high-power radio. Figure 42 shows that using the CC1000 radio for neighbor discovery in our MR-CAPM scheme reduces the energy consumption by 95%, while the CAPM scheme only achieves 75%.

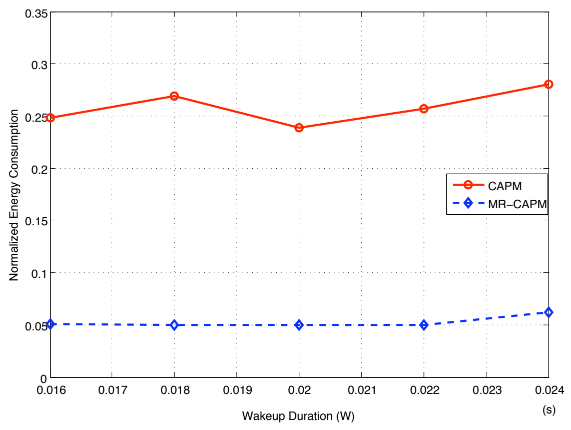


Figure 42: Normalized energy consumption using CC1000 radio for neighbor discovery in MR-CAPM

Though the energy savings make the CC1000 radio an attractive option for the MR-CAPM scheme, it is important to also understand the effects that the scheme has on the overall delivery ratio. From Figure 43, it becomes clear that using the CC1000 radio for neighbor discovery has a very high impact on the delivery ratio. This ratio especially become very poor once higher values for C and large distributed areas for nodes are used. However, this poor performance is only because the CC1000 radio has a very limited transmission range of 50m. Using MR-CAPM along with a low power radio (XTend) that has a transmission range of 250m, the same as the long range high power WaveLan radio, the MR-CAPM performs as well as the CAPM scheme in terms of the delivery ratio as shown in Figure 44

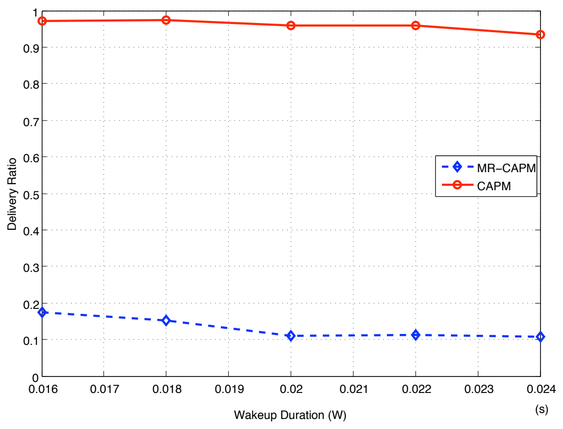
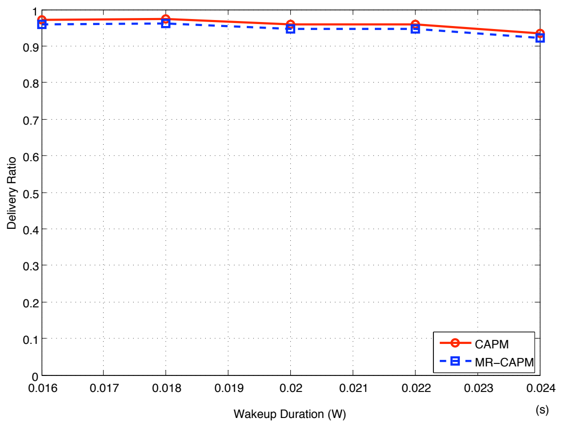
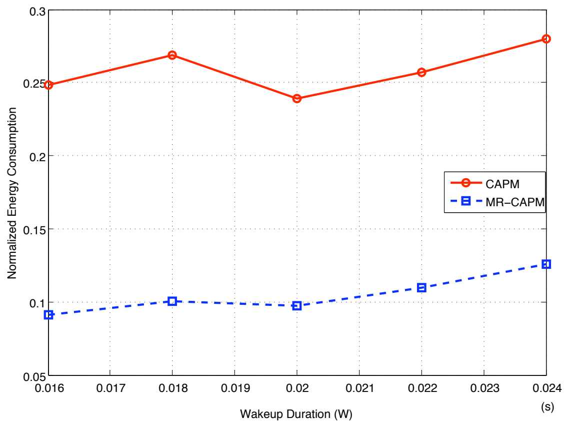


Figure 43: Delivery ratio using CC1000 radio for neighbor discovery in MR-CAPM



(a) Delivery Ratio



(b) Normalized Energy Consumption

Figure 44: Delivery ratio and normalized energy consumption using XTend radio for neighbor discovery in MR-CAPM

(a), while it outperforms the CAPM by reducing the energy consumption by half the power consumption as shown in Figure 44 (b).

From the previous results, it is clear that using the XTend radio for neighbor discovery in the MR-CAPM scheme is better than using the CC1000 radio. Therefore, all the following experiments use only the XTend radio as low-power radio for neighbor discovery. However, we adjusted the transmission range of the XTend radio to 250m since the transmission range for the low-power radio has to be less than or equal to the high-power radio (WaveLan 802.11) transmission range to avoid delivery failures as mentioned in Section 5.3.4.

Impact of node densities and traffic load

This experiment shows the effect of different node densities and traffic loads on MR-CAPM. We use the network scenario of 40 nodes distributed over 1000 x 1000 m^2, 2000 x 2000 m^2 and 3000 x 3000 m^2, and we use the same parameters that were used to evaluate the CAPM scheme as shown in Table 17.

Figure 46 shows the delivery ratio, the average delay, and the normalized energy consumption over different traffic loads and node densities. From Figure 46 (a) and (c), it is

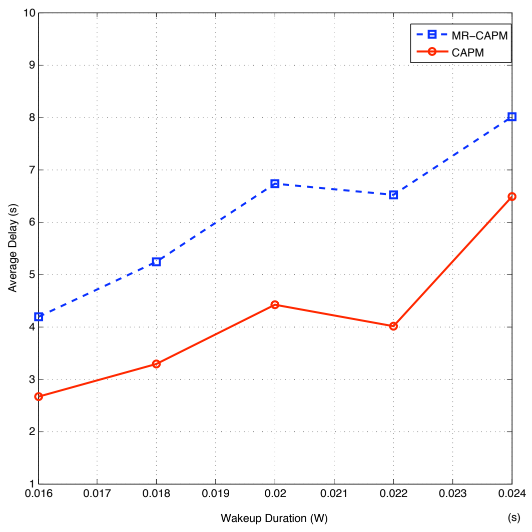


Figure 45: Average delay for both CAPM and MR-CAPM

Table 17: Parameter values

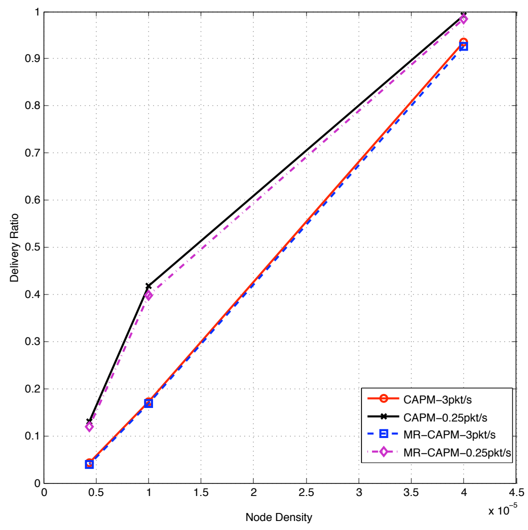
Packet rate	Node density	Sleeping pattern (W,C,K)
3pkt/s	1000 x 1000-40	(0.024, 1.67, 3)
	2000 x 2000-40	(0.01, 1,25, 4)
	3000 x 3000-40	(0.01, 1,6, 3)
0.25pkt/s	1000 x 1000-40	(0.04, 0.4, 12)
	2000 x 2000-40	(0.01, 0.4, 12)
	3000 x 3000-40	(0.01, 0.4, 12)

clear that the delivery ratio of the MR-CAPM scheme is comparable to the CAPM scheme, while the normalized energy consumption for the MR-CAPM is significantly less than the CAPM. The CAPM saves 85% energy at low load (0.25 pkts/s) and 75% energy at high load (3 pkts/s) over different node densities, while the MR-CAPM saves 92% energy at low load (0.25 pkts/s) and 90% energy at high load (3 pkts/s). Figure 46 (b) shows that the average delay of the MR-CAPM is comparable to the average delay of CAPM over different node densities and traffic load of 3 pkts/s, while its higher than the average delay of the CAPM at low load (0.25 pkts/s).

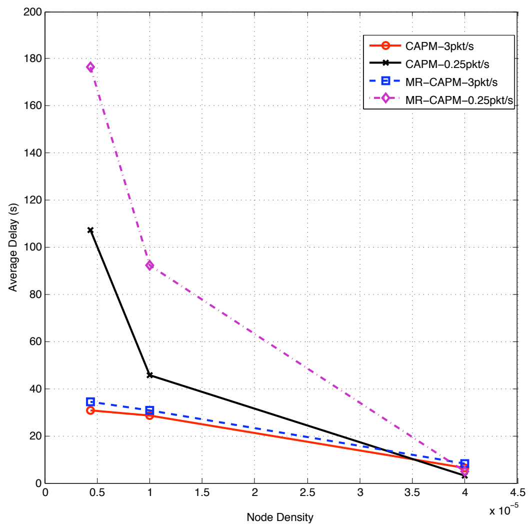
Figure 47 shows the delivery ratio, the average delay, and the normalized energy consumption vs the packet rate. This figure shows that the MR-CAPM can still achieve almost the same delivery ratio for different packet rates, while it consumes half the power of CAPM. Figure 47 also shows that increasing the packet rate will decrease the delivery ratio, while it increases the normalized energy consumption.

5.3.7 Conclusion

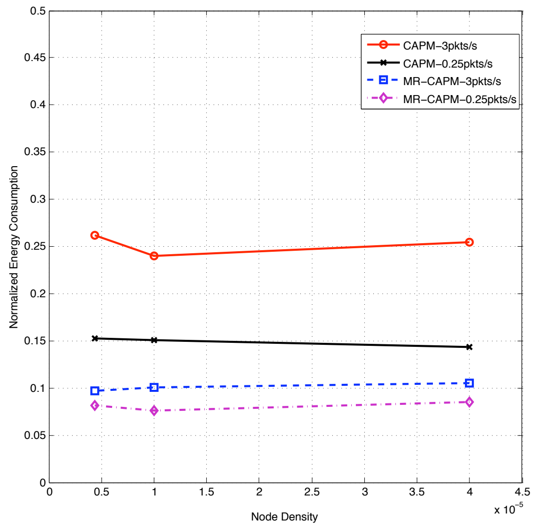
This work introduces a new power management scheme for sparse Disruption Tolerant Networks (DTNs). DTNs are characterized by frequent partitions, intermittent connectivity, and message delivery delays. Many of these networks suffer from resource constraints



(a) Delivery Ratio

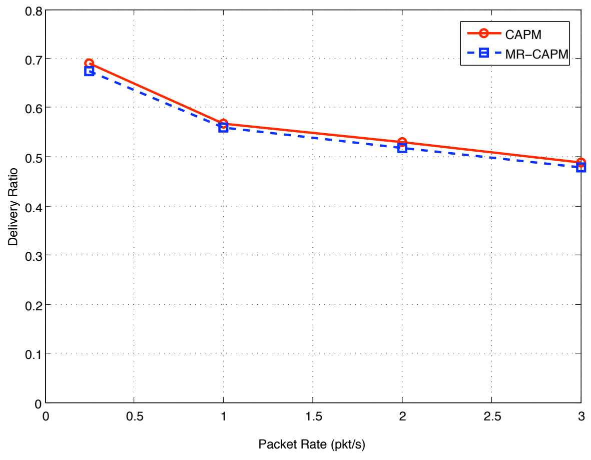


(b) Average Delay

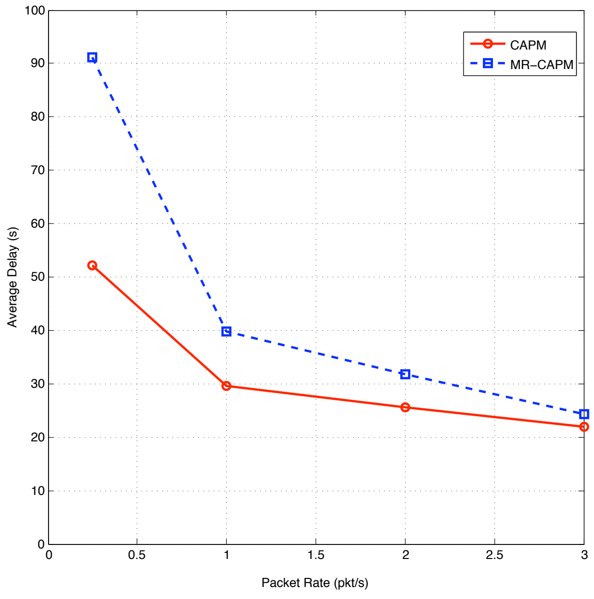


(c) Normalized Energy Consumption

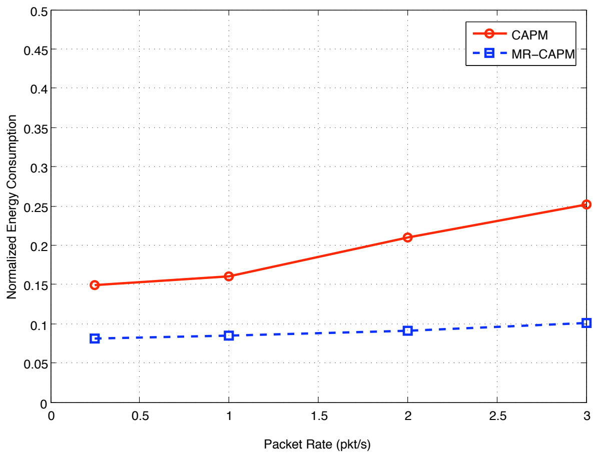
Figure 46: Impact of traffic load and node density on the delivery ratio, the average delay, and the normalized energy consumption for both CAPM and MR-CAPM



(a) Delivery Ratio



(b) Average Delay



(c) Normalized Energy Consumption

Figure 47: Comparison between MR-CAPM and CAPM (delivery ratio, average delay, normalized energy consumption) vs (packet rate)

and at the same time they are assumed to operate over a long period of time. Therefore, good power management approaches are needed to save energy and extend the network lifetime. Moreover, studies have shown that in DTNs, most of the energy is consumed while nodes search for contacts through their radio interfaces. These contacts are used as relays to forward the data to its destinations.

In this work, we present a new power management scheme for disruption tolerant networks called MR-CAPM. The MR-CAPM scheme is an asynchronous on-demand power management scheme which extends the CAPM scheme by implementing a low-power radio for neighbor discovery and high-power radio for data delivery. The simulation results obtained indicate that when a suitable radio is used for neighbor discovery, the MR-CAPM can reduce the energy consumption by 87% to 92% compared with the case without power management and by 55% to 68% compared with CAPM. Our results also show that MR-CAPM can achieve almost the same data delivery ratio of the CAPM scheme with half the power consumption. Our work highlights the need for choosing an appropriate radio since the choice not only effects the delivery ratio but the overall power consumption too. It is also important to bear in mind that the low-power radio must have always a maximum communication range that is equal to or less than that of the high-power radio so that only those nodes are discovered as neighbors to which data delivery is actually possible. The obtained results strongly argue in favor of using the MR-CAPM scheme and warrant further testing with other DTN routing protocols as well.

6 Security Management Infrastructure (SeMI)

An end-to-end secure communication in an inter-domain and heterogeneous environment is a key yet complex factor. Users have become critically dependent on information, and on the systems and networks used to provide access to this information. Protecting classified and sensitive information (by assuring integrity, authenticity and authorised access to it) as well as protecting the infrastructure (networks, systems) conveying such information is of key importance. One of the most important management challenges is to assure a predefined security level over multi provider environments with dedicated communication infrastructures, security mechanisms, processes and policies. A common understanding of the security requirements and mechanisms to assure this is vital. Thus, it is necessary to overcome management islands, mostly realized by commercial providers, and to work towards a common security management infrastructure (SMI) to get a common understanding of the security services and mechanisms to realize the requested level of security.

As one of the key functions of such an SMI is the provisioning of secured communication channels, i.e. VPN tunnels. These protection is necessary for the user data but also for management data, in order to protect network devices from being hijacked. Since cryptographic operations can consume notable CPU resources, it is important to understand the impact of security protocols on the scalability of the network management plane. This is especially important as Generic security protocols such as TLS or SSH usually have been designed with a certain application in mind (TLS for HTTP over TCP, SSH for terminal emulation and file transfer over TCP). Meanwhile, these protocols are used for other purposes and features sometimes are missing or need to be adapted. DTLS for instance is a variant of TLS running over an unreliable datagram service.

6.1 Towards a Security Management Infrastructure

IT Security is a matter of paramount importance and today's networks feature a large number of different security solutions - often not interoperable, complex to manage and laboriously to change or modify. This mostly leads to stovepipe systems with less flexibility and increased security concerns. Security Management Infrastructure (SMI) is an emerging research area that aims at assisting organizations in managing their security capabilities consistently and in provisioning security functions to organization entities. In this section we introduce a multi-layer architecture for a SMI system. It integrates various inhomogeneous security devices and services of an organization and provides a uniform interface for accessing them. Thus the SMI system establishes the basis for a global and consistent management of the security infrastructure according to organizational goals. To the best of our knowledge, it is the first such SMI approach and is currently under ongoing development.

6.1.1 Introduction

Security of IT assets is of ever increasing importance for most businesses as well as non-governmental and governmental organizations. Especially the military needs to secure its

IT infrastructure in order to protect its data as well as to ensure its operational state.

Therefore current network environments have to incorporate an ever-increasing variety of security services, devices, processes, and protocols. Especially in large military organizations, including joint and combined components, security measures were often implemented step-by-step or established only on demand, if mission requires, as "point-solutions" [199]. They solve one specific task, sometimes only for one dedicated user group, not looking how the chosen solution fits in the security architecture of the organization. The resulting environments are complex because the different security pieces from a diverse range of manufactures are usually not compatible.

Therefore, many of these organizations operate on an inhomogeneous and non-interoperable security infrastructure, which leads to stovepipe systems, and application- and task-specific "security silos" [177]. Applications using those security capabilities are implementing different standards or proprietary protocols and have to be changed if security components are replaced. Furthermore each "silo" or "stovepipe" must be configured individually - a laborious and error prone task. Moving to a more homogenous infrastructure is, however, mostly not possible because of limited funds, or not existing or not certified components.

The situation might still be overcome with the help of an overarching security architecture integrating security capabilities and managing them according to a global security policy [200]. Such an architecture is also required to provide the flexibility necessary in future dynamic military operations [201]. This overarching security architecture is termed Security Management Infrastructure (SMI) and is expected to solve these challenges [202].

Therefore a SMI needs to provide two main functions: (1) an organization-wide security middleware incorporating the inhomogeneous security devices and services and providing a single, transparent and implementation-independent security functions interface for being used by enterprise applications, and (2) an also uniform management interface for this middleware and the incorporated security services and devices [199]. The uniform management interface allows configuring and managing information from these security services and devices in an efficient and consistent manner and thus enhancing the organization's security.

Having these challenges in mind SMI can be defined as following. A Security Management Infrastructure (SMI), also known as Enterprise Security Management (ESM) [175], is an infrastructure providing unified access to the security capabilities of an organization. It, too, comprises the systems and resources required to order, create, disseminate, modify, suspend, and terminate the management controls, which are necessary to provide and operate security services, devices, and processes across the organization according to a security guideline [176], [202]. SMI is characterized by many heterogeneous, reused, and flexible security capabilities of different granularity and in different lifecycles within and across organizational boundaries. According to [176], [202] and [203] SMI capabilities include, e.g., Identity Management, Privilege Management, Metadata Management, Policy Management and Cryptographic Key Management. These security capabilities enable and manages basic security functions, such as perimeter defense, confidentiality, virus protection, protection of data at rest, or encapsulation of data during transmission [204].

Within this section, we introduce a multi-layer architecture for a SMI system, which integrates the various inhomogeneous security capabilities of an organization and provides a uniform SMI API for usage by the organization entities. This establishes the basis for a global and consistent management of the security infrastructure according to organizational goals and requirements.

6.1.2 Scenario

This section describes the IT security infrastructure of a newly set up joint and combined task force for a stabilizing mission. It should assist the local government of the host nation in establishing a secure and stable environment. The mission also includes external partners, which help accomplishing the mission goal (e.g., armed forces or police of the host nation, or non-government organizations such as humanitarian groups, which permanently or temporally support the mission).

Part of the task force is an allied headquarter, which operates a communication and information infrastructure. This IT infrastructure also includes various security devices, services, and processes to manage IT security - subsequently termed security capabilities. For purposes of our work (1) a security service is a piece of software considered independently and which can be composed of various sub-services (e.g., Credential management service may include a service to create X.509 based certificates), (2) a security device is hardware with software running on it (e.g., a firewall or VPN gateway appliance) and (3) a security process builds upon security functions of these services and devices and provides more complex functions (e.g., an authorization process with policy checking) [202], [205].

External partners are allowed to use specified resources of that infrastructure under the terms of a mission security policy. The security capabilities comprise multiple different security processes, services, and devices from various manufactures that have been tailored to specific mission tasks. Each capability, in its own way, performs the security functions that it was designed for. However, taken as a whole, the security services and devices are usually not compatible and often do not "talk to each other" [199]. Moreover various security operators, analysts, planners, and managers of security capabilities all maintain their own perspective and issues [206].

The scenario is visualized in Figure 48. It shows in the left half the communication and information infrastructure in the headquarter, which is protected by several security services (Security Service 1 to n, e.g., user account databases or a PKI), security devices (Security Device 1 to n, e.g., firewall or VPN appliances) and several security processes building upon the services and devices. The security infrastructure also controls the access of external partners.

The interrelationships of the different security capabilities should be depicted by analysing a sample security process - an access request of a user in the role of a mission planer who wants to gather best practice solutions from archived information of previous military missions (see right half of the figure).

Before the user is granted access to the secured database he needs to authenticate his identity. In this example the user has an X.509 certificate including the appropriate private and public keys on a smartcard. The provided identity reference is double checked with a LDAP server in order to confirm whether the identity is still valid. Between the device of the

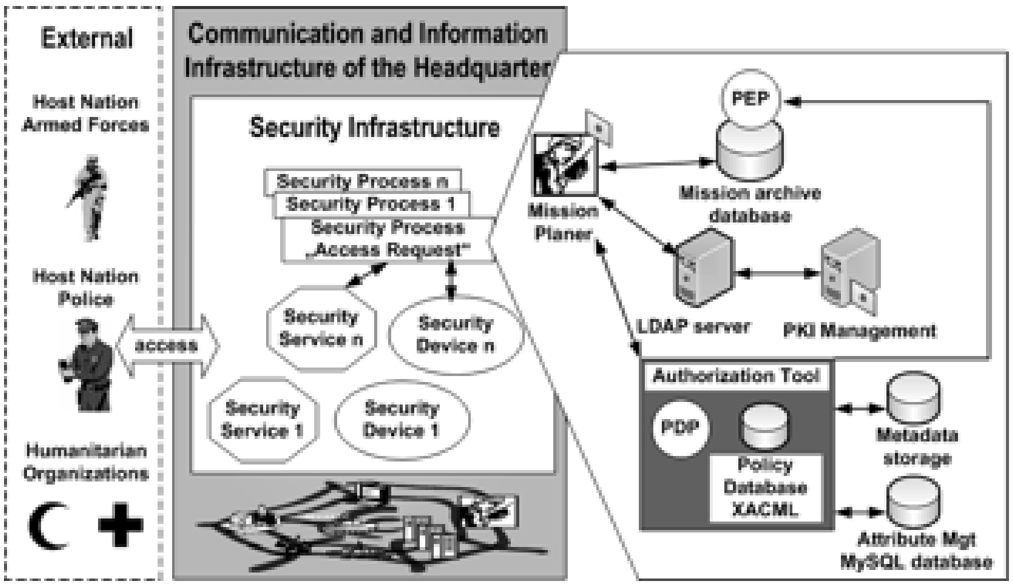


Figure 48: Security infrastructure with a joint and combined task force.

user (e.g., computer, PDA) and the authorization tool a SAML (Security Assertion Markup Language) assertion is created, which implements role-based Access Control policies defined in XACML (eXtensible Access Control Markup Language). Beside the information within the SAML assertion, the authorization tool requests additional metadata about the secured database and afterwards calculates the corresponding access control policies for the access decision. Furthermore user attributes are queried from a MySQL attribute management database.

All this information is taken by the Policy Decision Point (PDP) to decide on the access request of the mission planer, which is then forwarded to the Policy Enforcement Point (PEP) implemented in the mission archive database.

Access requests to data by external partners must be handled differently, if they have not been issued a smartcard with a certificate of a trusted certification authority. Furthermore, not all applications might rely on an external PDP for an authorization decision but implement authorization as a core component. In this case they will also store the authorization profiles in their own proprietary form by themselves.

If components of the security process and used protocols change (e.g., the LDAP server is replaced by an Active Directory server) applications and configurations at various places in the network may need to be changed.

A consistent configuration of all security processes can be accomplished by today only in manpower-intensive manual configuration steps performed by various security operators, administrators, and analysts [202].

Especially as military organizations of the future will rely on operational agility as a fundamental strategy in dealing with any types of adversary [201], systems and configurations may need to be adapted more frequently and in very short notice (e.g., if a nation suddenly requests to leave the task force because of internal political affairs).

6.1.3 Requirement Analysis

This section defines the requirements of an unified and collaborative Security Management Infrastructure, based on but not limited to the scenario described in the previous section.

First of all, the SMI needs to provide a single uniform interface to all security services and devices implemented in the organization. All persons and applications of the organization can make use of this interface, without knowledge of which concrete device and service are providing the information at the end. The SMI should be able to provide security functions for all kinds of internal or external entities including human or non-human users like standalone, two-tier, three-tier, and Web applications.

Therefore it is necessary that the SMI is able to accommodate all security services and devices an organization may operate in order to foster the move from manual to automated, or when technically feasible and operationally viable, to automatic security management operations [202]. Thus comprise especially the support of security capabilities, which are not yet or cannot be fully web enabled, e.g., legacy metadata file storage. As long as there is no support for such legacy systems using (propriety) protocols, conventional stand-alone security management and security function provisioning is required.

As the number of security services and devices, on the one hand, and the number of potential users of an SMI, on the other hand, can get quite high in big or federated organizations, a scalable architecture is required.

The number of security services and devices managed by one SMI must be changeable during the SMI's life time. Additional services and devices may need to be configured if the organization grows or if components are replaced. But also the set of supported devices and services must not be static. Continuously, new types of security services and devices evolve and manufactures are extending their product portfolio. Thus an open, extensible, and flexible architecture is necessary, which can easily accommodate new types of services and devices as well as a changed set of configured services and devices.

If the set of services and devices supported is changed, most likely the SMI API, providing the uniform interface to all security capabilities, must be changed, too. Therefore this interface must be flexible and extensible, too. Furthermore, new kinds of applications may have additional security requirements, which are not yet taken care of in the interface.

The security processes build upon the functionalities provided by the security services and devices. Their definition heavily depends on the security policy of the organizations (e.g., authentication requirements, authorization rules, or encryption policies) but also on the mission needs (e.g., providing access to data for first responders in case of a terror attack). Consequently, if the security policy of the organization changes, the SMI must support adapting the security processes easily and during run time.

As shown in the above scenario, additional complexity arises, if different organizations cooperate (e.g., in a joint task force, or when using outsourced security capabilities). Therefore inter-SMI data exchange is necessary, where SMIs of two or more organizations share information, e.g., for providing secure end-to-end communication. The use of standardized and non-proprietary protocols to communicate and exchange information between security capabilities will support this inter-organization share of information.

The management of the SMI has two aspects: (1) the management of the operative data, i.e., the management of all security capabilities, and (2) the management and configuration of the SMI system itself.

A SMI system will allow a central management of all operative security data. Therefore the SMI API needs to provide an interface to all security-related data stored in the concrete security services and devices (e.g., user accounts, password, or security policies). By concept of a SMI, the interface must be independent of the services and devices used. However, this degree of independency might not be achievable for every configuration function. For managing the underlying security services and devices direct access to their management interfaces will still be necessary, respectively to support separation of concerns within the security management staff.

For configuration and adaption the SMI system itself, another set of management functions are necessary. SMI management must be able to monitor and change the behaviour of the SMI middleware, including the security processes and which concrete security services and devices to incorporate. This comprises the management and support of any security capabilities along their full life cycle - planning, provisioning, operation, modification and withdrawal.

6.1.4 Related Work

This section is structured into two parts. First we provide some theoretical foundations concerning security management models and general design principles. Secondly, we present an overview of current SMI approaches and exchange standards.

Security management is defined as one system management functional area of FCAPS within the OSI management architecture (ISO 10164) [200]. Here security management functions are described generically in order to be implemented by security management tools. The ISO 17799 Part 2 (2002) established the code-of-practice and the specifications of an Information Security Management System (ISMS) [203], which presents a methodology for providing and managing security services. It provides guidelines on how a management framework for enterprise security should be implemented. ISO/IEC 27001 (Nov. 2005) has been prepared to reemphasize the code-of practice of ISO 17799 with few amendments and additions of controls that will enhance and improve the ISMS further [204]. A framework for IT service management as unified basis for the development of further concepts for service management is proposed in [205].

A Service Oriented Architecture (SOA) packages functionality as interoperable services. This allows different applications to exchange data with one another. [207] presents a Service Oriented Security Architecture (SOSA) as a collection of security services forming a security infrastructure used by Web service providers. In addition the Data Centric Security Model (DCSM) abstracts security services (e.g., authentication and authorization) and their underlying mechanisms into interfaces that directly support central data management policies [204]. The principal of an agent-based hierarchical SMI architecture is mentioned in [199].

Different activities in military, industry, and governmental organizations regarding unified interfaces to and management of security services, devices, and processes can be observed [199]. None of the activities, however, is flexible and holistic enough to be capable of ensuring the required level of interoperability and flexibility for various security capabilities.

The current SMI tools that focus on managing the entire security equipments of an organisation are trying to solve the problem with an overarching security management that supports the security administrator, operators and planers in managing their heterogeneous systems. These systems thus integrate the different security processes and components only on the management layer. Even these new kinds of security management systems assist security personnel greatly; still (1) lots of configuration work, especially when interconnecting security mechanisms and components, need to be done manually, and (2) the range of supported security fields is often limited. Such commercially available systems include Symantec ESM 6.5 [208], IBM's Tivoli Secure Way suite [209], ArcSight's ESM [210], CA's eTrust SCC [211], BMC Control-SA [212], e-Security's OeSP [213] and HP's OpenView [214].

Beside these solutions, there are some standards and approaches for specific security areas. In [215] a Web-based security management system providing security management services for small and medium businesses through Application Service Providers is introduced.

For the exchange of authentication and authorization data, standards as OASIS SAML [216], specifications of Liberty Alliance and the Web Services Federation Language [217] are implemented with their main focus on Web-based services [218]. Furthermore in the security area of crypto key management the Key Management Interoperability Protocol (KMIP) [219] can be used. Security Services Markup Language (S2ML) [220] is aimed at creating a common language for sharing security information about transactions and end users between businesses engaged in online B2B and B2C transactions.

Implementation-independent security services are provided by the Generic Security Services Application Program Interface (GSS-API). It defines some application programming interfaces for accessing security services [221]. Limitations of the GSS-API include that it standardizes only authentication, and not authorization, and that it assumes a client-server architecture.

To summarize the related work analysis, we can identify that an integrating security management architecture, including solutions for the addressed requirements is missing so far. In addition relevant design principles have to be adopted and enhanced from the mentioned models.

6.1.5 Design of a SMI Architecture

In this section an architecture for a SMI is designed. The main challenge during this design process is the number of different possible concrete security services and devices, which have to be accommodated by the SMI. The security services and devices may differ in two dimensions:

- First, how a function of a service or device may be interfaced with. Each service or device may have its own protocol for communication, e.g., the Light-weight Directory Access Protocol (LDAP), a proprietary API, or Simple Object Access Protocol (SOAP) in case of a Web service.

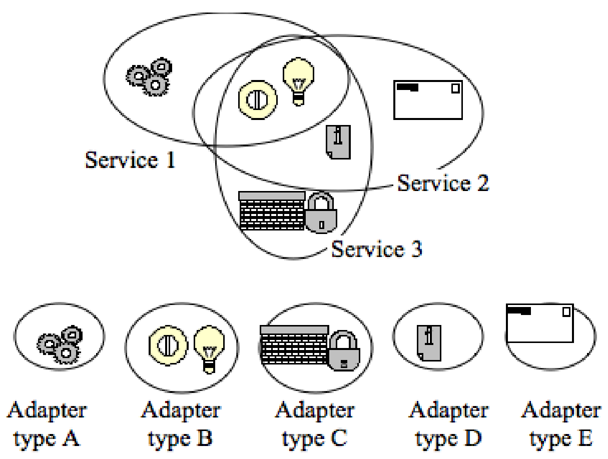


Figure 49: Decomposition of service sets in adapter types.

- Second, the function set a concrete service or device provides. This set might be very different from service to service and from device to device. Furthermore the sets might mutually overlap.

This problem is solved by an abstraction and service decomposition layer, called Adapter Layer. It consists of different types of adapters, where each type may have a number of service- or device-dependent implementations. Each adapter has two interfaces: a generic interface, which is common for all adapters of one type, and a service- or device-dependent interface, which implements the (proprietary) interface of the concrete security service or device. A sample adapter type might comprise functions for changing firewall rules. Different adapters for, e.g., Linux netfilter, Checkpoint Firewall 1, or Microsoft Internet Security and Acceleration Server might be implemented.

For service decomposition it is required that the set of functions being implemented by the different types of adapters has to be disjunct. Furthermore, as a rule of thumb, the function set of one type of adapter should be chosen so that a concrete service or device either implements all functions of an adapter or none (Violation of this rule should be avoided but are not problematic). Each type of adapter can have multiple instances within the SMI system.

Figure 49 visualizes the decomposition of functionality sets in Adapter types where each function is shown as an icon. It shows in the upper half three services, e.g., providing identity management related functions. The service sets of the different services are not identical but they overlap - some functions are provided by all three services, some only by two, and some even only by one. As a first step during SMI implementation, Adapter types need to be defined with disjunct function sets and with service-independent function definitions (lower half of the figure).

In this example Service 1 provides functions from Adapter type A and B, Service 2 from Adapter types B, D, and E and Service 3 from Adapter types B, C, and D.

In a next step service-dependent adapters need to be implemented. In the shown case a service 1-dependent implementation of Adapter type A and B would be required as well as a service 2-dependent adapter implementation of Adapter types B, D, and E, and Adapter types B, C, and D need to be implemented in accordance to Service 3.

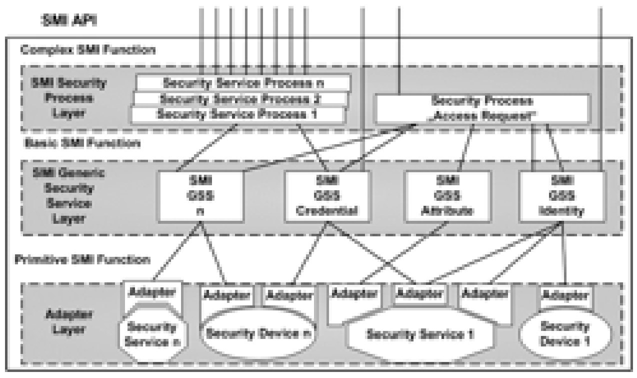


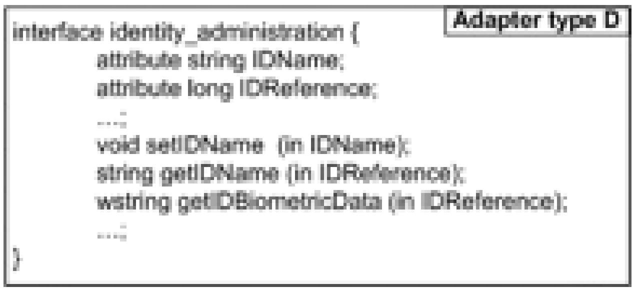
Figure 50: Multi-layer architecture for a SMI system.

The reason for this service decomposition is reducing complexity in the next higher layer and the number of Adapter types. Looking at the functions provided by for example Adapter type D, those are not implemented by all security services but by all services providing an adapter of type D. Therefore Service 2 and Service 3 can be treated as type D services from layers residing above the Adapter Layer.

Figure 50 shows the complete architecture of the SMI. The Adapter Layer is on the lowest layer of the SMI, including the adapters and the concrete security services and devices. One or more adapters map their function set to the next layer, providing a uniform interface consisting of so called Primitive SMI functions to the Generic Security Service Layer (GSSL). In Figure 51 a simplified generic interface of an Adapter type relating to the field of Identity Management is visualized. This interface is described with a formal description approach as for example Interface Description Language (IDL).

The GSSL comprises a list of Generic Security Services (GSS). Each GSS composes different Primitive SMI Functions to Basic SMI Functions and provides location transparency for SMI data. Location transparency means that the GSS implement algorithms to find transparently requested data among different concrete security services and devices, and therefore hides the distributed storage of data from higher layers. A GSS represents one of the security fields such as Identity Management, Cryptographic Key Management or Attribute Management.

An example for the location transparency can be visualized by referring back to Figure 49: Be Service 2 and Service 3 two databases for user accounts; Service 2 is a Windows Active Directory for internal users and Service 3 a SQL database for external users. For a concrete request the respective GSS needs to choose the right or all databases based



```
interface identity_administration {                Adapter type D
        attribute string IDName;
        attribute long IDReference;
        ...;
        void setIDName  (in IDName);
        string getIDName (in IDReference);
        wstring getIDBiometricData (in IDReference);
        ...;
}
```

Figure 51: Generic interface of a sample adapter type.

on some criteria (e.g., username or function). If for example Adapter type D provides a search function, an appropriate GSS would need to query all services implementing an adapter of type D.

The set of Basic SMI Functions is the first part of the API the SMI system provides to users. The Basic SMI Functions, however, are used by the SMI Security Process Layer, too. As the name of the layer already suggests, it implements complex security processes or security workflows. The functions provided by a security process are termed Complex SMI Function and form the second part of the SMI API.

Also part of the SMI API are functions to manage the SMI and the security devices and services, as well as the security processes of the SMI Security Process Layer. An example of such a management function would be CreateIdentity, which creates a new identity reference in an appropriate security service. Based on parameters of a CreateIdentity request and the configuration of the SMI, the GSSL would forward this request to the right security service. In above example if creating an identity for an external user, the request would be forwarded to Service 3.

6.1.6 Future Work and Conclusion

SMI is a challenging task, mainly because of the underlying infrastructure characterized by many heterogeneous, reused, and flexible security capabilities of different granularity and in different life cycles within and across organizational boundaries.

In this work we introduced a multi-layer architecture for a SMI system. It integrates various inhomogeneous security devices and services of an organization and provides a uniform interface for accessing them. Thus it establishes the basis for a global and consistent management of the security infrastructure according to a common goal.

In our next steps we will focus on the description of the uniform interfaces of the three layers. For that we are analysing various security environments to decompose functionality sets of security services and devices in order to describe a set of adapters, before the generic interface for each of the adapter is defined. Thereafter the resulting set of primitive functions provided by the adapter generic interfaces is clustered according to security areas to create Generic Security Services. Afterwards these services can be composed in a set of security processes.

Our future work will also include developing a prototype of a SMI system based on current security services, devices, processes and protocols running in the domain of the German Federal Armed Forces.

6.2 Introduction

The secure shell protocol (SSH) [222] is widely used to securely access command line interfaces of network devices and host systems. SSH establishes an encrypted tunnel between a client and a server and allows applications to create several independent channels within the encrypted connection. An SSH server is authenticated by using the server's public/private key pair, also known as the server's hostkey. Clients are authenticated using one of several client authentication methods. The most widely used client authentication methods are the public key, password, and keyboard-interactive authentication methods.

The success of SSH as the protocol of choice to securely access command line interfaces has led to the adoption of SSH as a protocol for other programmatic network management interfaces, namely NETCONF [223, 224] and SNMP [225, 226]. Using the same protocol to securely access interactive management interfaces as well as more programmatic management interfaces reduces the operational costs associated with key management.

The SSH protocol computes new session keys whenever a new SSH session is established. A commonly used SSH key exchange mechanism is based on the Diffie-Hellman algorithm, which is computationally expensive. This introduces significant latency and high processor load when short-lived sessions are established frequently, especially on low-end devices such as DSL routers or wireless access points.

An experimental study of the performance of SNMP over SSH [227, 228] has shown that the overhead associated with SSH session establishment is significant. On a slow machine (Sun Ultra Sparc IIi), an increase of latency by a factor of 15 has been observed for the SSH session establishment and authentication procedure. A significant portion of the delay is caused by the key exchange procedure, i.e., the computation of the session keys. Since many existing SNMP-based management applications and ad hoc scripts do not maintain long lived management sessions, the adoption of SSH as a technology to secure programmatic management interfaces can have significant impact on the CPU load of managed devices and also on the scalability of management applications.

In order to address this problem, we propose a session resumption feature for SSH allowing clients to resume sessions without having to compute new session keys. Session resumption significantly reduces the latency and computational overhead associated with the establishment of SSH session and thus improves the scalability of management applications and reduces the costs of introducing strong security on low-end devices.

The rest of the paper is structured as follows. A short review of SSH is provided in Section 6.3 before we present the proposed session resumption mechanisms in Section 6.4. Section 6.5 discusses some implementation details and Section 6.6 provides an evaluation of the session resumption extension. Related work is discussed in Section 6.7 before we conclude the paper in Section 6.8.

6.3 Secure Shell Protocol

The secure shell protocol (SSH) has a layered architecture [222] consisting of three major components:

- The SSH transport protocol [229] is used to establish a secure communication channel between a server and a client. It provides functions to negotiate message authentication code (MAC), compression, and encryption algorithms, to authenticate the server to the client and to establish session keys.
- The SSH user authentication protocol [230] authenticates the client to the server. It runs over the SSH transport protocol and supports several client authentication

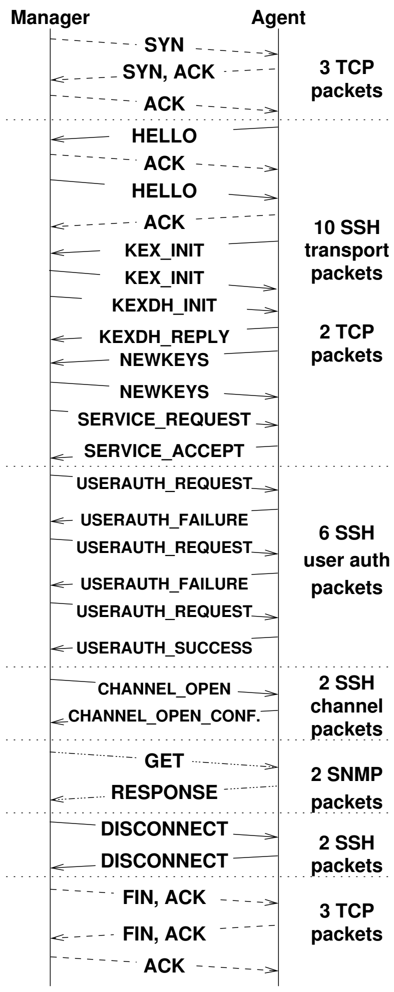


Figure 52: IP packets exchanged by executing an SNMPv3 `GET` operation over SSH

methods, such as shared passwords, public keys, or keyboard-interactive (challenge response).

- The SSH connection protocol [231] multiplexes the encrypted connection into several independent logical channels.

A typical SSH exchange carrying a single SNMP `GET` request has been captured on a local area network and is shown in Fig. 52. After the client has established a TCP connection, the client and the server execute the SSH transport protocol. The first messages exchanged are hello messages identifying among other things the SSH protocol version the client and the server are using.

The next two SSH messages start the key exchange procedure. The server and the client exchange the list of key exchange, server authentication, encryption, MAC, and compression algorithms they support using `SSH_MSG_KEXINIT` messages. The client and the server select an algorithm and then start the selected key exchange. A key exchange algorithm produces a shared secret K and an exchange hash H. Encryption and authentication keys are derived from K and H.

A widely used key exchange algorithm is the Diffie-Hellman key exchange. The client sends a `SSH_MSG_KEXDH_INIT` message, which just includes the client's Diffie-Hellman value

e. The server returns its value f in a SSH_MSG_KEXDH_REPLY message. In addition, the server signs his reply using his private hostkey. The client then verifies the server's hostkey, usually against a cached copy of the server's public key.

After executing the Diffie-Hellman exchange, the server and the client exchange SSH_MSG_NEWKEYS messages to inform each other that they from now on use the session keys established by the executed key exchange algorithm. From this point on, all communication is encrypted and authenticated using the negotiated algorithms. For security reasons, session keys are renegotiated after a given amount of time has passed or a given amount of data has been transferred. All key negotiations for a session are carried out using the same key exchange algorithm.

In the last step of the SSH transport protocol exchange, the client sends a SSH_MSG_SERVICE_REQUEST message to request a specific service, usually the SSH user authentication service. The client executes the user authentication protocol by iterating through the supported user authentication methods. The first SSH_MSG_USERAUTH_REQUEST usually fails, causing the server to return the list of supported user authentication methods in the SSH_MSG_USERAUTH_FAILURE reply. The client then iterates through the supported methods until a positive reply (SSH_MSG_USERAUTH_SUCCESS) has been received or all methods have been tried.

Using the SSH connection protocol, the client requests to open a channel by sending a SSH_MSG_CHANNEL_OPEN message. The server then responds with a SSH_MSG_ CHANNEL_OPEN_CONFIRMATION message.

Fig. 52 shows the exchange of SNMP GET / RESPONSE messages over the newly created SSH channel before the SSH connection is closed by sending SSH_MSG_DISCONNECT messages, followed by a TCP connection tear-down exchange. Note that TCP ACKs are piggybacked in our example after the initial hello exchange.

6.4 Session Resumption

Session resumption requires that session state — established keys and selected algorithms among other things — is kept for some time after an SSH session has ended.

Clients must certainly maintain their own session state to support session resumption. Server state, on the other hand, could be handled in two different ways. The first option is to let the server maintain its session state as discussed in Section 6.4.1. The second option is to have the client maintain the server's session state as discussed in Section 6.4.2.

6.4.1 Resumption using Server-Side State

Session resumption using server-side state is straightforward. The server must maintain a session state cache for recently closed SSH sessions. The client and the server indicate their willingness to perform session resumption by negotiating the use of a special session resumption key exchange algorithm. The standard key exchange algorithm negotiation process of SSH can be used for this purpose. If both sides choose to use the session resumption key exchange algorithm, the client sends the session identifier to the server in the first session resumption key exchange message (`SSH2_MSG_KEXSR_INIT`). The message also contains a MAC computed over the session keys. The server looks up the cached session and verifies the MAC. If successful, it returns an acknowledgement (`SSH2_MSG_KEX_SR_OK`), followed by a standard `SSH2_MSG_NEWKEYS` exchange. On failure, a `SSH2_MSG_KEX_SR_ERROR` is sent and the key exchange proceeds using another key exchange algorithm, or fails. If the session is long-lived, further key negotiations happen using a different key exchange algorithm. Every such negotiation invalidates any previously cached state for that session.

A drawback of this method is that the server has to maintain the session state cache. If a server has to deal with a very large number of clients, then the session state cache can grow out of proportion forcing the server to purge cache entries quickly, thereby reducing the number of successful session resumptions. This problem can be addressed by taking away responsibility for maintaining state from the server entirely, as described below.

6.4.2 Resumption using Client-Side State

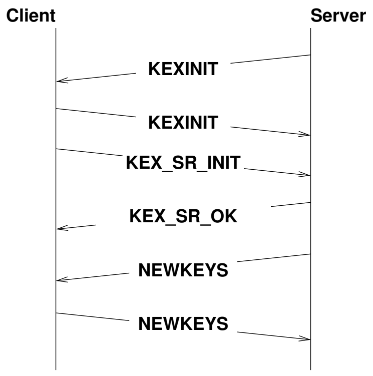
Session resumption using client-side state requires introducing a ticket. The ticket is created by the server and contains all information required by the server in order to restore the session state later. After every successful key negotiation that is not itself session resumption, the server generates a ticket, encrypts it, and hands it over to the client in a `SSH2_MSG_KEX_SR_TICKET` message.

The key used to encrypt a ticket is generated during server start-up, and is known only by the server. Therefore the client must keep state for the session to be resumed beside the ticket, and must be able to correctly associate that state with the corresponding encrypted ticket. If a resumed session is long-lived, further key negotiations happen using an algorithm other than session resumption. Each such renegotiation causes the server to generate and send a new ticket.

During session resumption, the client first sends a `SSH2_MSG_KEX_SR_INIT` message that contains the encrypted ticket and a MAC over the session identifier. (The identifier is known to the server and to the client, but not to third parties. It is never transmitted unencrypted, in contrast to the server-side state scenario.) The MAC is necessary to ensure that the client is indeed able to restore the session using that ticket, and to prevent an attacker from intercepting a ticket and using it as its own. If the server decrypts the ticket and verifies the MAC successfully, it sends a `SSH2_MSG_KEX_SR_OK` message, and both sides exchange `SSH2_MSG_NEWKEYS` messages as per the SSH protocol. On failure, a `SSH2_MSG_KEX_SR_ERROR` message indicates that both sides must fall back to another key exchange algorithm, or disconnect.

The structure of the ticket has been designed to only contain the information absolutely necessary to resume a session. Everything else, i.e., all data necessary for a session but not contained in the ticket, is derived during the key exchange procedure in the same way as it is derived for other key exchange algorithms.

A ticket consists of a `TicketContent` structure that is only visible to the server. Its encrypted representation is wrapped in `Ticket` structure, which contains information visible to everyone.



(a) key exchange

Figure 53: Session resumption key exchange

The `TicketContent` structure is shown on Fig. 54. The `session_id` field contains an identifier generated for each session, and `session_id_len` is its length. A resumed session has the same identifier as the one it was resumed from. For each communication direction (client to server and server to client), a session has one algorithm for encryption, one for authentication, and one for compression. These are kept in the next six fields. An encryption algorithm (`TicketEnc`) structure keeps the name of an encryption algorithm, its key, and its initialization vector (IV). The lengths of the key and IV, as well as of the authentication key below, are determined by the algorithm and do not need to be stored. An algorithm is only described by its name under the assumption that both the server and the client will map the name to the same algorithm - a reasonable expectation given that the two used that algorithm for a session not long ago. Similarly, a `TicketMac` structure keeps the name and key of a MAC algorithm. A compression algorithm requires no key, and is only indicated by its name. The `hostkey_type` field indicates the algorithm used to authenticate the server to the client. The server and client version strings provide a reasonable indication that the SSH server or resp. client software has not changed since a ticket was issued. All strings in this structure after `session_id` are NULL-terminated and consequently do not need to have their lengths stored.

A `TicketContent` structure contains very sensitive information and must be protected from third parties, but also from the client. Thus it must be encrypted with a key that is only known to the server. This key and its IV are generated at server start-up. The server is free to choose any encryption algorithm, and any MAC algorithm for ticket ID validation (described below).

What the server actually sends to the client is a `Ticket` structure, shown on Fig. 55. The `seq_nr` field is used by the server if it chooses to keep track of what tickets have been issued. It can then quickly discard some invalid tickets, without attempting decryption or ID validation. Note that `seq_nr` is not related to the SSH sequence numbers used for all packets. The `id` field serves as a unique identifier of that ticket. It is a MAC computed over the key used to encrypt the ticket, the sequence number, and the ticket time stamp. It ensures that the latter two have not been tampered with. The length of `id` is determined by the MAC algorithm in use. The `enc_ticket` field contains the encrypted `TicketContent`, and `enc_ticket_len` is its length. Finally, `time_stamp` indicates the time when the ticket was issued; tickets older than a given threshold can be discarded immediately. It is up to

```
struct TicketEnc {
    char* name;
    u_char* key;
    u_char *iv;
};

struct TicketMac {
    char* name;
    u_char* key;
};

struct TicketContent {
    u_char* session_id;
    u_int session_id_len;
    TicketEnc tenc_ctos;
    TicketEnc tenc_stoc;
    TicketMac tmac_ctos;
    TicketMac tmac_stoc;
    char* tcomp_ctos;
    char* tcomp_stoc;
    int hostkey_type;
    char* client_version_string;
    char* server_version_string;
};
```

Figure 54: `TicketContent` structure

```
struct Ticket {
    u_int seq_nr;
    u_char* id;
    u_char* enc_ticket;
    u_int enc_ticket_len;
    int64_t time_stamp;
};
```

Figure 55: `Ticket` structure

the server to decide how many times and for how long a ticket can be used.

6.4.3 Compatibility Considerations

Care must be taken that the SSH extension interoperates with clients and servers that do not support session resumption. This is achieved by using the key exchange negotiation mechanism of SSH. To support session resumption, we introduce several new key exchange messages. According to [232], the message numbers 30 to 49 are specific to the key exchange method in use and can thus be reused for session resumption key exchange messages [222]. However, in order to obtain session keys, additional messages must be introduced. This requires to go through an IETF consensus process in order to allocated well known message numbers. In the meantime, there is a private number space that can be used for experimentation.

A server supporting client-side state session resumption might send a ticket to a client that does not support tickets. According to the provisions of [229], this should lead to a `SSH_MSG_UNIMPLEMENTED` message and not to a failure.

6.4.4 Security Considerations

Server and client implementations must take care to properly protect their session state caches and/or tickets. This is especially important to clients that must store tickets persistently between executions.

In the server-side state scenario, the session keys are never transmitted, because both sides are assumed to know them. If a third party captures the transmitted session identifier and attempts session resumption using the captures session identifier, the third party will not have the session keys required to produce the correct MAC to validate session resumption.

For client-side state session resumption, session keys are encrypted with a key only known to the server. The client is assumed to know the content of the encrypted ticket already. Non-encrypted portions of the ticket such as the ticket sequence number and timestamp are protected from tampering by the ticket identifier, which is a MAC that requires the server-only key to compute. A third party that captures a ticket would not know the session identifier, because it is encrypted in the ticket. The third party will not be able to produce the correct MAC over the ticket in order to validate the ticket. Even if it could somehow guess the session identifier, it would still not be able to resume the session, because the session keys contained in the ticket are also encrypted.

Because neither side determines any session keys that the other side will use, replay and man in the middle attacks are impossible. Session resumption only affects the SSH transport layer and only restores the encryption and authentication keys used to protect communication from third parties. Both the server and the client must still authenticate each other separately even if a session was just resumed.

6.4.5 Mobility Considerations

The ticket itself does not contain any information related to network identifiers such as IP addresses or port numbers. As such, the ticket itself is not bound to any specific network attachment points and it should be possible to resume SSH sessions even if a client changes its network attachment point in between. Whether it is feasible to resume SSH sessions where servers change their network attachment point depends on the details of the host authentication and which information is used by the parties involved to identify the server's hostkey.

6.5 Implementation

The session resumption extension described in the previous section has been implemented based on the OpenSSH implementation version 4.7p1. We have implemented session resumption with client-side state. The OpenSSH code-base is separated into three major parts: a static library, a server executable, and a client executable. Session resumption is implemented as a key exchange algorithm in both the server (`srs.c`) and client (`src.c`), with the majority of code shared between them, and therefore placed within the library (`sr.c`). One of our goals has been to minimize the amount of changes necessary to existing OpenSSH source code. Thanks to SSH's layered design, we only had to modify the SSH transport layer implementation, without ever looking at the user authentication or connection layers. Our current implementation consists of about one thousand lines of code of which only 10% are additions or modifications to existing source files.

Normally, the SSH algorithm negotiation procedure described in [229] is used to select an algorithm for key exchange. It is assumed that the algorithm itself cannot fail as long as the underlying transport does not fail, and that it will be used again for long-lived sessions without further negotiation. The most significant change to OpenSSH we had to make is the possibility of graceful failure of a key exchange algorithm, e.g., if an invalid ticket has been used. Upon such failure, both the server and client restart the key exchange with whatever algorithm would have been selected if session-resumption were not available. This fallback procedure does not require the exchange of any additional network packets. It is also used to reset the key exchange algorithm after successful resumption so that long-lived sessions can periodically renegotiate new session keys.

OpenSSH uses a privilege separation model, described in [233]. In this model, for every connection the SSH server daemon forks a process called a ”monitor”, which runs with full privileges because it must be able to authenticate users. The monitor itself forks unprivileged child processes, which handle all communication with the client. The monitor contains all sensitive information a server must know, and carries out operations that require special privileges on behalf of the child. At any time, there is a set of requests a child is allowed to make; any forbidden request immediately terminates the session. If, through some security exploit, an attacker compromises the server he is communicating with (the unprivileged child), he will not be able to gain privileges on the server machine, because the child that was compromised does not have them.

We made minor modifications to the set of allowed requests, in response to what we believe are minor mistakes in the OpenSSH implementation. These issues had never been triggered before, because all key exchange algorithms other than session-resumption work in essentially the same way. These, along with other minor OpenSSH issues we found, have been reported to the developers.

We note that our current implementation of session resumption does not conform to the privilege separation model, because we give the unprivileged children the secret key used to sign tickets, instead of giving it to monitors which must then use it on behalf of the children. We did this only as a shortcut to save time and we fully intend to fix this problem; there is no fundamental design limitation that causes the issue.

Serialization for both ticket transfer and storage is done using OpenSSH's `buffer` API. It differs from a straightforward copy of the memory structure only by prepending the length of a string before each string.

The client currently saves a received ticket and the corresponding session keys in two temporary files. For the ticket this is not a problem, but storing the session keys in this way is very insecure. Anyone who obtains them can use them to decrypt a resumed session.

Ticket management is a client's responsibility and is not part of the SSH protocol or of our extension. Our basic ticket storage is sufficient as a proof of concept, but it must not be used in a production environment.

We have identified the following important improvements that our current implementation requires:

- Foremost is proper conformance to the privilege separation model. Unprivileged children must not have access to the secret key used to sign tickets.
- Transferring tickets from a server to a client must be made optional and configurable. For example, the server should not try sending tickets if the version string of the client indicates lack of session resumption support, or the client does not advertise session resumption in its set of key exchange algorithms.
- Reasonably configurable secure ticket management for session resumption clients would allow using session resumption in a production environment.
- We must update our implementation against the latest OpenSSH version, which is 5.1 at the time of writing.

Once all of these issues are addressed, we will propose session resumption for inclusion into mainline OpenSSH.

6.6 Evaluation

We have evaluated the session resumption mechanism with client side state by running the command `ssh $host exit` and measuring the overall execution time from the beginning of the SSH exchange until the SSH session has been closed down. The experiments were performed on three Debian GNU/Linux machines (see Table 18). The machines were connected via a switched Gigabit Ethernet with sufficient capacity. The machine called `veggie` was running the SSH client while `meat` and `turtle` acted as SSH servers.

Table 18: Machines used during the measurements

Name	CPUs	RAM	Ethernet	Kernel
`meat`	2 Xeon 3 GHz	2 GB	1 Gbps	2.6.16.29
`veggie`	2 Xeon 3 GHz	1 GB	1 Gbps	2.6.16.29
`turtle`	1 Ultra Sparc IIi	128 MB	100 Mbps	2.6.20

Starting an interactive session using OpenSSH is a relatively expensive operation involving potentially complex things such as creating several processes, password validation, setting up of a shell and a user's environment. The absolute numbers are thus not comparable to other uses of SSH as a secure transport where the overhead of establishing the execution environment is much smaller (e.g., SNMP over SSH).

During the measurements, we used the hash function HMAC-MD5, the AES-128 encryption algorithm, and the RSA algorithm with a key size of 1024 bits as the public key cryptosystem.

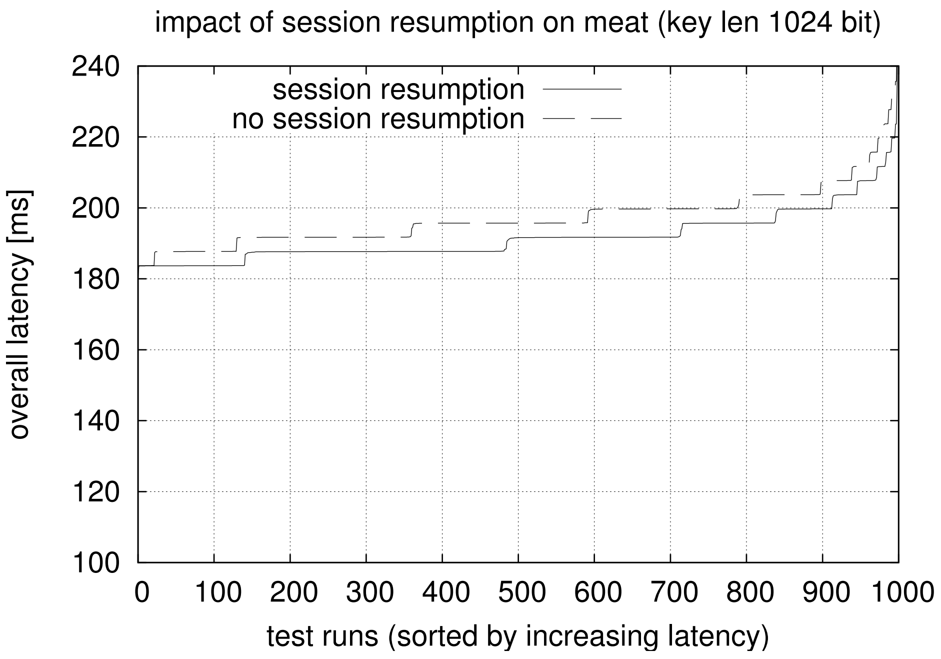


Figure 56: Latency of 1000 ssh sessions with and without session resumption on a fast machine (`meat`)

Fig. 56 shows the latency we measured on the fast server `meat`. We sorted the 1000 data points by the measured latency. The area between the two curves indicates the overall time saving achieved in the 1000 iterations. On `meat`, $5.4ms$ have been saved on average. Fig. 57 shows the latency we measured on the slow server `turtle`. On `turtle`, $310.4ms$ have been saved on average. Obviously, the impact of the slower CPU on the latency is significant and the effect of session resumption becomes much more visible.

It should be noted that we used a key size of 1024 bits, which is still considered secure but likely breakable in a few years. With larger key sizes (2048 bits or 3072 bits), the gains of session resumption will also be visible on faster machines, as shown in Fig. 58. Note that the performance of session resumption is not impacted by the RSA key length.

6.7 Related Work

Session resumption is a well known concept of the Transport Layer Security (TLS) protocol [234]. The core TLS specification supports session resumption with server-side state. Shachan et al. [235] introduced TLS session resumption with client-side state, which has meanwhile been standardized in [236]. Using self-signed tickets is a well known technique for re-authentication in authentication protocols [237].

The positive effects of TLS session resumption have been studied in [238, 239, 240]. These studies report that the key exchange has a major impact on the performance of TLS and they confirm that session resumption is an effective tool to reduce overhead.

Teemu Koponen et al. [241] extended the SSH and TLS protocols to support resilient connections that can span several sequential TCP connections. They introduce a new

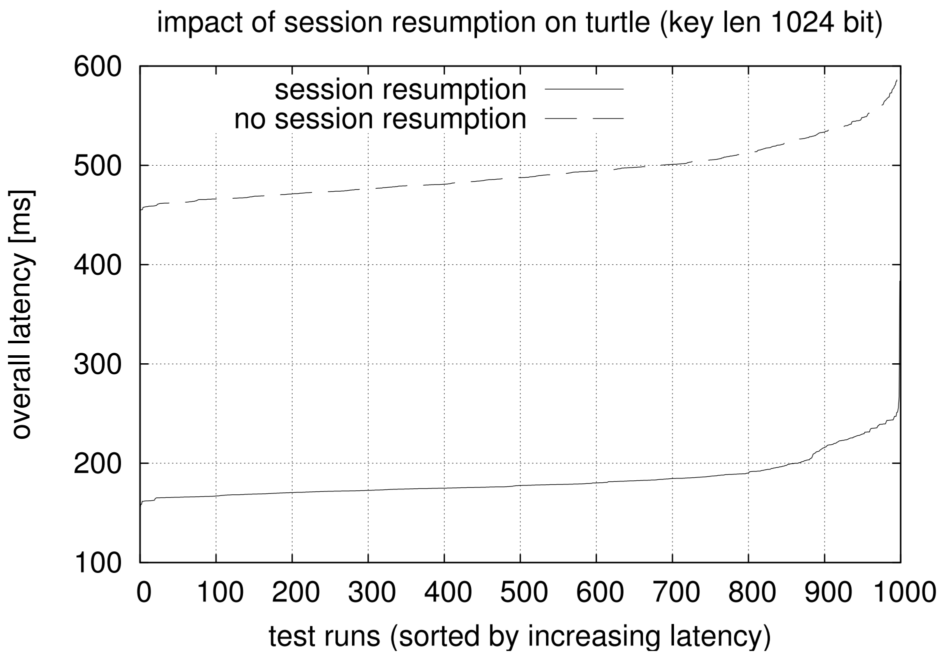


Figure 57: Latency of 1000 ssh sessions with and without session resumption on a slow machine (`turtle`)

key exchange method for SSH called 'resilient', which is similar to our session resumption mechanism with server-side state described in Section 6.4.1. They do not provide a mechanism to resume sessions with client-side state. Another difference is that we do not aim at restoring a session by resynchronizing buffers or compression/encryption state. Instead, our focus is just to restore the session keys of a previously used session.

The OpenSSH implementation supports a feature called connection sharing. With connection sharing enabled, an SSH client connected to a certainer server can act as a master. Other SSH clients (called slaves) running on the master's host connecting to the same server can ask the master to tunnel the new connection through the master's SSH connection. From the slave's point of view, this is very fast since no key exchange or authentication exchange is needed. Of course, this only works if the slave and master share a trust relationship and a long lasting SSH connection can be maintained.

6.8 Conclusions

The increasing usage of SSH for non-interactive and short-lived sessions requires to take a critical look at its performance. Because early HTTP versions required closing connections to indicate the end of documents, TLS has been optimized for short-lived sessions from the very beginning.

Unlike TLS, SSH has been designed with clear layer separation in mind, at the expense of message complexity and a larger number of round-trips. One significant factor impacting the SSH session establishment performance is the initial session key exchange. The latency introduced by the key exchange algorithm (typically Diffie-Hellman) can be significant, especially on the slow processors commonly used by consumer devices such as

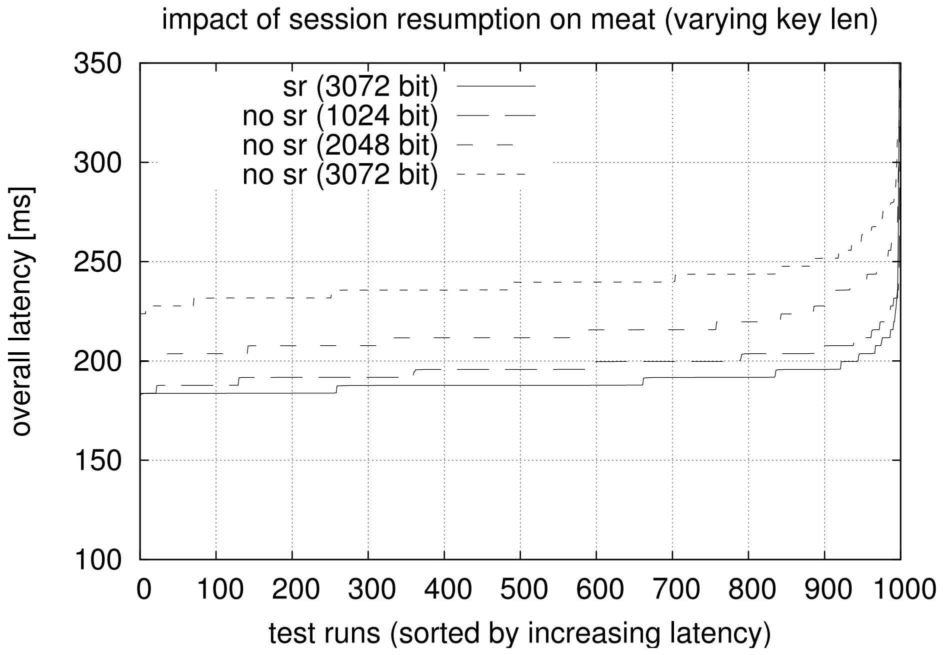


Figure 58: Latency of 1000 ssh sessions with and without session resumption on a fast machine (`meat`), varying the key length

wireless access points or DSL routers.

We address this problem by proposing an SSH extension allowing applications to resume previous SSH sessions. The server's session state can be kept either on the server itself, or on the client in the form of a ticket. The solution is backwards compatible and can therefore be deployed gradually.

7 Virtualization Monitoring (VirtMon)

7.1 Introduction

Virtualization is called to be a key technology in the future Internet. In fact, virtual devices and virtual networks allow for a "splitting" of the network to allow better operation mechanisms. Nevertheless, this paradigm is not exempt of penalties because the virtualization process imposes overheads that slowdown the network processes. Then, a careful analysis has to be done before applying the technique to get an appropriate trade-off between benefits and drawbacks. In addition, a key requirement is to find appropriate monitoring mechanisms for the virtualized resources and networks. The Virtmon project is addressing these challenges.

VirtMon Project is structured around four activities, namely monitoring system approach, identification of scenarios and evaluation network setup, and extension to EmanicsLab. This final report summarizes the achievements so far. Section 7.2 presents a generic Virtual Monitoring Architecture that states the grounds of monitoring of any kind of virtual devices. Section 7.3 is devoted to a practical implementation of monitoring of computational resources in virtual routers by means of SNMP and common system tools. The virtualization mechanism is adopted is Xen and the operative system is basic Linux. This section includes the performance data obtained through the testing of a virtualization scenario. Finally, Section 7.4 outlines the monitoring approach adopted in EmanicsLab, the virtual laboratory setup by EMANICS.

7.2 Virtual Machine Monitoring Architecture

We have defined an architecture that is designed around the concept of producers and consumers. That is, there are producers of monitoring data, which collect data from probes in the system, and there are consumers of monitoring data, which read the measurements. The producers and the consumers are connected via a network which can distribute the measurements that are collected. The architecture itself defines Probes and Data Sources which act as the sources of measurement data, and it defines Data Consumers which collect those measurements. This approach is a key aspect of an information management infrastructure also described in this section.

7.2.1 Overview

In many systems probes are only used to collect data for system management. However, to increase the power and flexibility of the monitoring, given the dynamic nature of virtual machines, we introduce the concept of a Data Source. A data source represents an interaction and control point within the system that encapsulates one or more probes. A probe can send a well defined set of attributes and values to a consumer at a pre-defined interval. The goal for the monitoring system is to have fully dynamic data sources, in which each can have multiple probes, with each probe returning its own data. The data sources will be able to turn on and turn off probes, or change their sending rate. Ideally

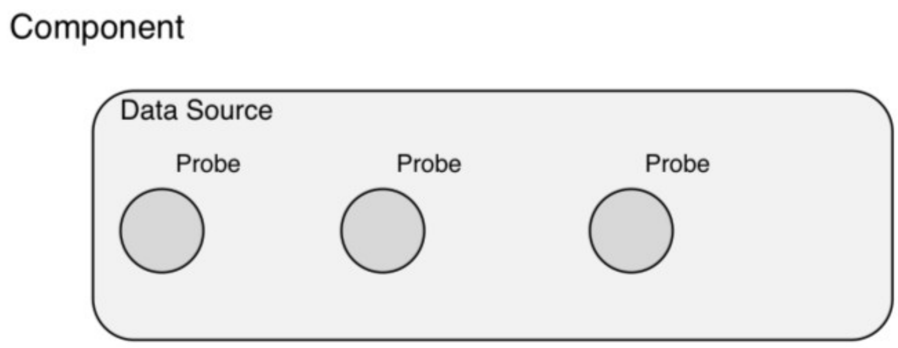


Figure 59: Abstraction of a component with its monitoring data source

the data source implementation will have the capability for reprogramming the probes on-the-fly. In this way, it will be possible to make probes send new data if it is required. For example, they can send extra attributes as part of the measurement. A further useful facility for a data source will be the ability to add new probes to a data source at run-time. By using this approach we will be able to instrument components of the system without having to restart them in order to get new information. Such an approach for data sources and probes is important because the monitoring requirements of virtual machines need to grow and adapt for new management requirements over time. If the monitoring system is a fixed point then the management will be limited. Figure 59 shows a data source with some probes.

The probes can be written to collect any kind of data. Also, rather than re-implement everything from scratch, it is clearly beneficial to interface with existing monitoring frameworks in order to collect data. To fit in with the concept of data source and probe, they can be encapsulated with the relevant adapter in the implementation.

7.2.2 Implementation

We have developed a first working implementation. This implementation acts as a foundation for doing distributed monitoring and provides the framework which binds all of the producers and consumers together.

We have written probes which provide information on the many physical hosts that are used as a platform for running a virtual machine hypervisor. The hypervisor is the core component of many cloud computing environments and is used to run multiple virtual execution environments (VEEs). As well as writing probes to gather data from physical machines, we have evaluated how the distributed monitoring framework can interact with the hypervisor in order to extend the monitoring capabilities to include virtual machines running within the hypervisor. Our work in this area has tried to address the flexible and adaptable requirements of cloud computing and how these fit in with libvirt (the library that provides access to various hypervisors through an abstraction layer). We have made progress in some areas, but have been undermined by the unstable nature of some parts of the libvirt implementation. We are currently investigating how to overcome these issues in order to create a stable hypervisor monitor.

The current design and implementation for monitoring virtual machines is dependent on the fact that a virtual machine can be created, can execute, and can shutdown at run-time whilst the monitoring is still running. In essence, the hypervisor presents a collection of machines that change over time. From the consumers' point of view, it is important that the virtualized machine should look like an individual host, so we ensure that there is one probe sending data for each virtual machine. The alternative is to have one probe for the whole hypervisor, but using this approach then every measurement will contain data for every virtual machine currently executing on the hypervisor. The consumers will not see individual hosts, but a collection of them. Consequently, virtualized hosts will have to be treated differently from real hosts, which is not a desirable situation.

To ensure that there is one Probe per virtual host, and to accommodate the dynamic nature of the hypervisor we have architected a solution shown in Figure 2. In the top part of the diagram we can see the hypervisor and the virtualized execution environments (VEEs). To get data from the hypervisor there is a HypervisorController which gets a list of running VEEs from the hypervisor, on a regular basis. The Hypervisor Controller compares the current list of VEEs with the list retrieved last time. From this it determines (a) if there is a new VEE, in which case it asks the HypervisorDataSource to add a new HypervisorProbe for that VEE, or (b) if a VEE has shutdown, in which case it asks the HypervisorDataSource to delete the HypervisorProbe for that VEE.

As stated earlier, a data source represents an interaction and control point within the system, so it is the data source that is responsible for adding and deleting the probes and also to collect any measurement data from the probes and to pass it onto the network for the consumers. In this case, the HypervisorProbes each run independently of each other, in their own thread, and collect data regarding their assigned VEE at a regular interval. The data collected is structured that same as data collected from a real physical host, such as CPU and memory usage.

The final element of Figure 60 is one that provides single threaded access to the hypervisor by using locks. This is strictly necessary as the libvirt implementation is only capable of doing one request at a time, but has no locking or serialization mechanisms of its own. In our multi-threaded implementation, using libvirt directly can cause libvirt to fail.

Data collection from virtual machines can be carried out in two ways:

- Separate measurement streams can be sent for each of the virtual hosts on a physical machine. This is particularly useful in situations where the actual location of the virtual host is not important, and also ensures that measurement data from one virtual host is not coupled to a physical machine. The data streams are independent, and so consumers just see streams of measurements from virtual hosts and physical hosts in the same way because the structure of the measurements is identical in both cases.

- The data for all of the virtual hosts is collected into a table and sent as one stream from the hypervisor. This is useful where the consumer needs a snapshot of the current state of all the virtual hosts on a physical machine. The consumer is responsible for unpacking data for each virtual host from the table.

In summary, the hypervisor monitor we have designed and built can collect data from the virtual execution environments in a dynamic and adaptable way. It can collect information about CPU, memory, and network usage.

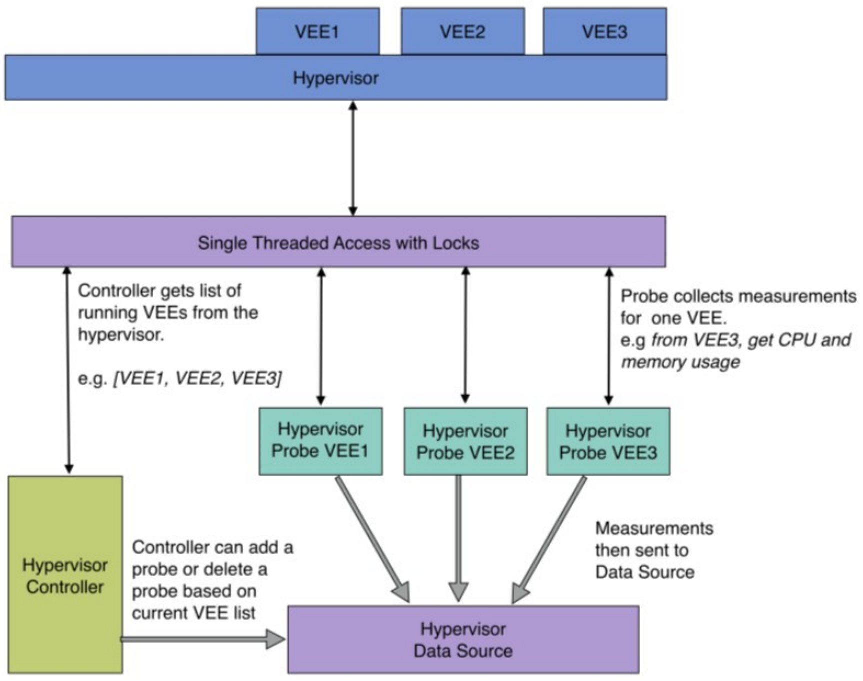

VEE1
VEE2
VEE3
Hypervisor
Single Threaded Access with Locks
Controller gets list of running VEEs from the hypervisor.
e.g. [VEE1, VEE2, VEE3]
Probe collects measurements for one VEE.
e.g from VEE3, get CPU and memory usage
Hypervisor Probe VEE1
Hypervisor Probe VEE2
Hypervisor Probe VEE3
Hypervisor Controller
Controller can add a probe or delete a probe based on current VEE list
Measurements then sent to Data Source
Hypervisor Data Source

Figure 60: Monitoring of multiple virtual environments

7.2.3 Scenario: Information Management Infrastructure

A key issue in the management of emerging networks is the development of a common control space, which has autonomic characteristics and enables heterogeneous network technologies, applications, and network elements to interoperate efficiently. Management applications need to be adaptive to a rapidly changing environment with respect to specific network properties, and service or user requirements, as examples. This implies that management applications and network entities should be supported by a platform that collects, processes, and disseminates information characterizing the underlying network. These applications are commonly implemented using virtual execution environments in order to gain the flexibility required to be adaptable to the changing network conditions. Consequently, we need to be able to monitor these virtual environments to gather the relevant metrics for information management.

By having an increased awareness of the properties and state of the network, we can bridge the gap between high-level management goals and the configuration that achieves them. In this respect, we consider an infrastructure that manages both information flow and processing within the network as an important stepping-stone towards this objective.

Key design requirements of an information management infrastructure are: (i) information collection from the sources (e.g., network devices), (ii) information processing that produces different information abstractions, and (iii) information dissemination to the entities that exploit that information. It is common that such design approaches aggregate information using aggregate functions. Consequently, real-time monitoring of network parameters may introduce significant communication overhead, especially for the root-level nodes of the aggregation trees. Information flow should adapt to both the information management requirements and the constraints of the network environment, one example

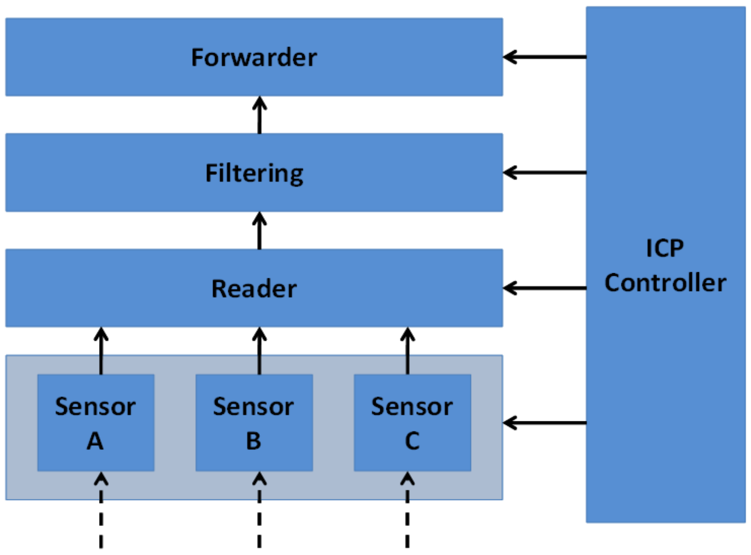


Figure 61: Structure of Information Collection Points

being: changes in the information collection configuration.

To this end, we consider an Information Management Overlay (IMO) [242], which is a management infrastructure that collects, processes, and disseminates network and system information from/to the network entities at real-time, acting as an enabler for self-management functionality. The virtual machine monitoring approach developed is a key component of the information management platform.

7.2.4 Information Management Overlay Components

This section describes in detail the two main components of the information management overlay: Information Collection Points (ICPs) and as Information Aggregation Points (IAPs).

Information Collection Points

Information Collection Points act as sources of information: they monitor hardware and software for their state, present their capabilities, or collect configuration parameters. Three types of monitoring queries to an ICP are supported: (i) 1-time queries, which collect information that can be considered static, e.g., the number of CPUs, (ii) N-time queries, which collect information periodically, and (iii) continuous queries that monitor information in an on-going manner. ICPs should be located near the corresponding sources of information in order to reduce management overhead. Filtering rules based on accuracy objectives should be applied at the ICPs, especially for the N-time and continuous queries, for the same reason. Figure 61 shows the structure of an ICP, which we have designed and implemented, consisting of 5 main components: the sensors, a reader, a filter, a forwarder and an ICP controller. These are described below.

The sensors can retrieve any information required. This can include common operations such as getting the state of a server with its CPU or memory usage, getting the state of a network interface by collecting the number of packets and number of bytes coming in and out, or getting the state of disks on a system presenting the total volume, free space, and used space. In our implementation, each sensor runs in its own thread allowing each one to collect data at different rates and also having the ability to turn them on and off if they are not needed.

The reader collects the raw measurement data from all of the sensors of an ICP. The collection can be done at a regular interval or as an event from the sensor itself. The reader collects data from many sensors and converts the raw data into a common measurement object used in the IMO framework. The format contains meta-data about the sensor and the time of day, and it contains the retrieved data from the sensor.

The filter takes measurements from the reader and can filter them out before they are sent on to the forwarder. Using this mechanism it is possible to reduce the volume of measurements from the ICP by only sending values that are significantly different from previous measurements. For example, if a 5% filter is set, then only measurements that differ from the previous measurement by 5% will be passed on. By using filtering [243] in this way, the ICP produces less load on the network. In our case, the filtering percentage matches the accuracy objective of the management application requesting the information.

The forwarder sends the measurements onto the network. The common measurement object is encoded into a network amenable measurement format. The measurements are encoded using XDR [244] as a way to minimize the size of the transmitted data. The XDR format is commonly used in monitoring systems [245] in order to reduce network loading.

The ICP Controller controls and manages the other ICP components. It controls (i) the lifecycle of the sensors, being able to turn them on and off, and to set the rate at which they collect data; (ii) the filtering process, by changing the filter or adapting an existing filter; (iii) the forwarder, by changing the attributes of the network (such as IP address and port) that the ICP is connected to.

We have developed various sensors, which can measure attributes from CPU, memory, and network components of virtualized hosts by interacting with a hypervisor to collect these values. Finally, we have sensors that can send emulated measurements. These are useful for testing and evaluation purposes, with one example being an emulated response time.

Information Aggregation Points

IAPs apply aggregation functions to the collected measurement information. The aggregation process increases the level of information abstraction, thereby transforming the data into a structured form, but at the same time reducing the load on the network. Aggregation works in situations where information consumers do not need a continuous stream of data from an ICP, but can get by with an approximation of the data. For example, getting an occasional measurement with the average of the volume of traffic on a network link may be enough for some applications. Some common aggregation functions include SUM, AVG, STDDEV, MIN and MAX.

Figure 62 shows the structure of an IAP, which we have designed and built, consisting of 7 main components: a collector, an aggregation specifier, a selector, an aggregator, a filter, a forwarder and an IAP Controller. All of these components are described below.

The collector collects measurement data from the network and converts the XDR encoded measurement format into a measurement object. After this the measurement objects are

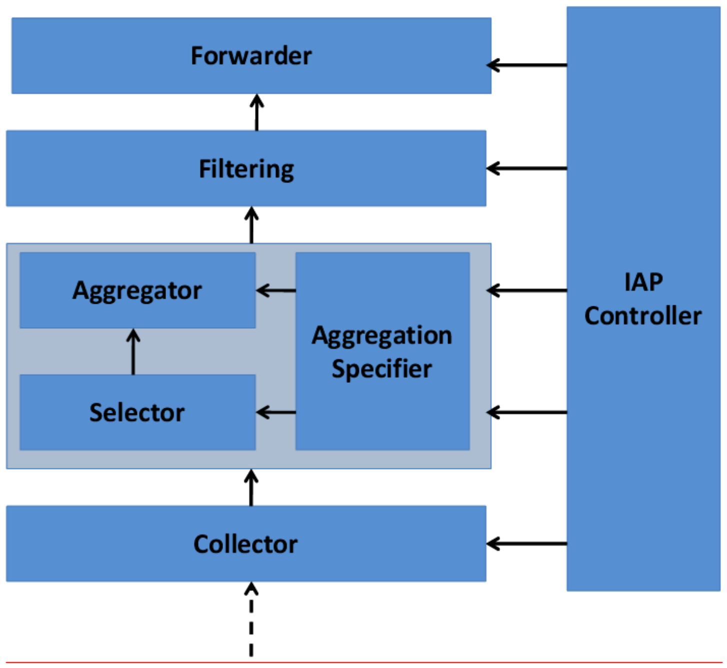


Figure 62: Structure of Information Aggregation Points

saved in a data store for later aggregation processing. The data store used in the IMO is the Timeindexing Framework [246] which allows any kind of data to be stored and retrieved using timestamps or time intervals. The Timeindexing Framework provides the mechanism by which arbitrary sequences of measurements can be selected and aggregated. It creates an index into the data, called a timeindex, and provides an API for accessing the data.

The aggregation specifier, the selector, and the aggregator are actually combined into an aggregation engine. The aggregation specifier specifies when the aggregator executes, what it aggregates, and how it aggregates. These three specifications are similar to those used in SLA compliance systems [247], because the process of analyzing the data is similar.

The when specification is of the form: wake up every N seconds, which will cause the aggregation engine to wake up regularly to provide an aggregation. The what specification takes the form of a time interval, such as ”from now, back 30 seconds”. The how specification is the name of a function to aggregate the data, such as AVERAGE, SUM, etc.

The selector selects the required measurements to aggregate using the what specification. It determines what data is eventually chosen by applying the time interval, such as ”from now, back 30 seconds”, to the timeindex and selecting the relevant measurements. In this case, it will cause the selector to select the most recent 30 seconds worth of data. As the data store uses timeindexing, the time interval can be changed arbitrarily. Once the selection is complete the selected data is passed to the aggregator.

The aggregator aggregates the selected measurements presented by the selector. It uses the how specification to aggregate data. Although it is most common to use aggregation functions, such as SUM, AVERAGE, STDEV, MIN and MAX, the IAP Controller can pass in an arbitrary function into the aggregator in order to do the aggregation. This gives considerable power and flexibility when determining aggregations. Once the aggregation is calculated, the aggregated measurement data is passed to the filter. The filter takes measurements from the aggregator and can filter them out before they are sent on to the forwarder. Again, this reduces the volume of measurements by only sending values that are significantly different from previous measurements. Using filtering in this way in the IAP, like filtering in the ICP, less load is produced on the network.

The forwarder sends the aggregated measurements onto the network. The common measurement object is encoded into the same network amenable format as in the ICP, the XDR. By having the same network format, the consumers of the measurement data do not need to know if data has come directly from and ICP or has come from an IAP. This allows hierarchies of elements to be composed as an IAP can further aggregate data from other IAPs, if this is required.

The IAP Controller controls and manages the other IAP components. It controls (i) the collector, by changing the attributes of the network that the IAP listens to, (ii) the aggregation process, by managing the aggregation engine and by passing in the aggregation specifier, (iii) the filtering process, by changing the filter or adapting an existing filter, (iv) the forwarder, by changing the attributes of the network (e.g. IP address and port) that the IAP sends to.

The aggregation engine itself is flexible enough to be given different aggregation specifications by the IAP Controller in order to process the data in varying way. For example, it can be configured to wake up once an hour and select data for the last day, and then apply an aggregation function. This is achieved using a mechanism that relies on plugins. These plugins represent code blocks, which can be pre-defined, such as an average aggregator, or can be defined to suit the need.

Information dissemination is done through the IAPs. As an application may request a specific piece of information from an IAP, the deployment location of the IAPs should also consider the locations and the traffic requirements of the nodes retrieving information. As well as requesting information, an application has the option to subscribe to an event-based notification service by setting an appropriate threshold to a specific type of information. Whenever this threshold is exceeded, the application is notified.

7.2.5 Evaluation

In this section, we describe the results of experiments carried out to evaluate different aspects of the IMO infrastructure:

- The filtering mechanism, where we show how information accuracy can be traded for communication cost
- The information source, where we show the impact of different sources of information, including CPU and memory sensors, and an emulated source (i.e., the response time emulator)

Experimental Setup

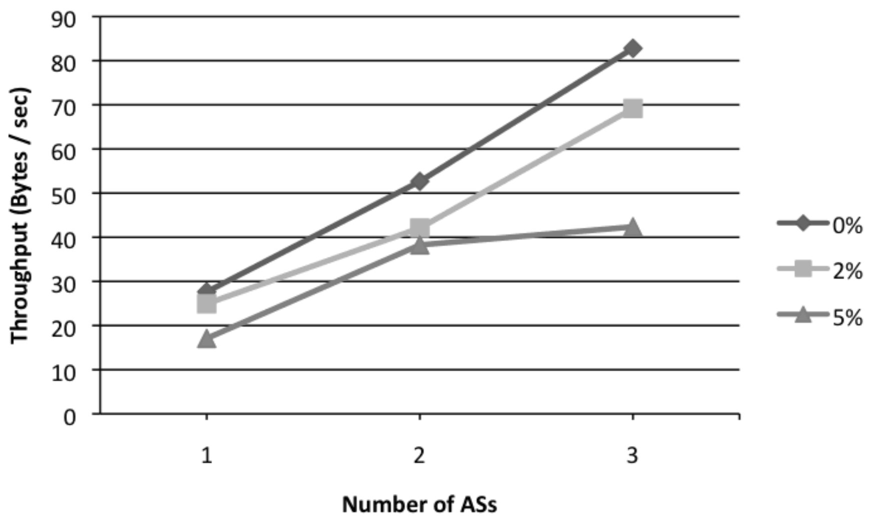


Figure 63: Impact of Filtering at the Aggregation Point

We deployed the IMO infrastructure on a test-bed that consists of two SUN servers with 4 CPU cores (2.5 GHz each) and 8GB of memory and one with 8 CPU cores (1.9 GHz each) and 32GB of memory. Each server hosts 10 virtual machines using the XEN virtualization platform [248]. With such a setup, we emulated a network that consists of 33 machines. In our experiments, each physical machine acts as an emulated Autonomous System (AS).

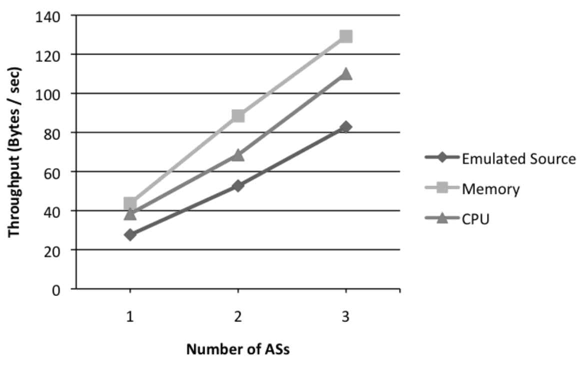
The number of physical machines was gradually increased from one to three, in order to show the behavior of the system in different topology scales. In these experiments, all the virtual machine monitors act as information sources. Throughput is measured at the IAPs and transmission time from the information collectors to the Information Aggregation Points. The Information Collection Points and Information Aggregation Points transmit a measurement every 30 seconds.

Experimental Results

We evaluated the statistical accuracy of our results by performing 10 runs and calculating the standard deviation. We did not observe any large deviations. For example, in a scenario with 2 ASs and 20 VMs acting as information sources, the standard deviation in the throughput measurements was just 3.4% of the average throughput value. The evaluation results are presented below.

Impact of Information Filtering: As we can see in Figure 63, the throughput at the AP is reduced significantly due to the filtering algorithm. So, filtering can adjust the trade-off between information accuracy and communication overhead. The impact increases with more information sources and larger topologies.

Impact of Information Sources: In this scenario, we show the impact of different information sources on the throughput and transmission time. We use different information sensors that monitor CPU utilization and memory usage. Additionally, we use a sensor that collects measurements from a response-time emulator. As shown in Figure 64 (right), the type of information has an impact on the communication cost. In our case, the emulated source associates with much less throughput than the CPU and memory sources. This is not reflected to the transmission time (Figure 64 left), in the case of 3 ASs.



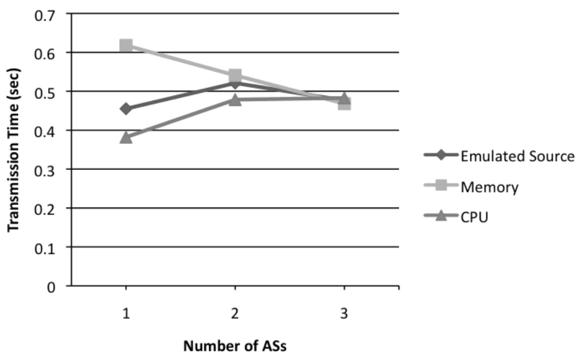
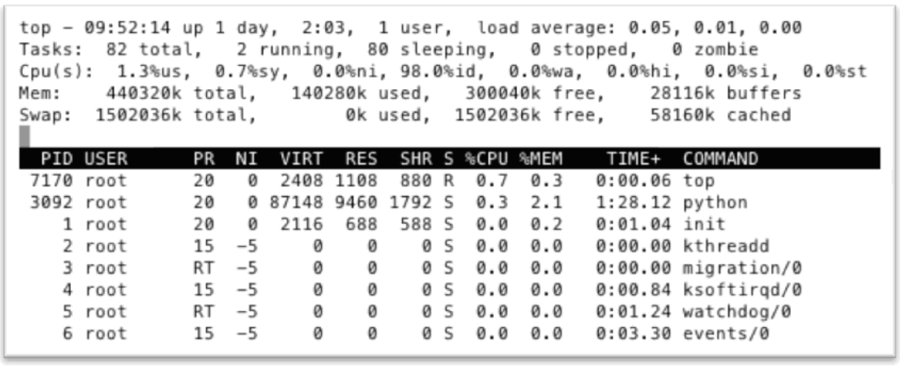


Figure 64: Impact of Information Source at the Aggregation Point



```
top - 09:52:14 up 1 day,  2:03,  1 user,  load average: 0.05, 0.01, 0.00
Tasks:  82 total,   2 running,  80 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.3%us,  0.7%sy,  0.0%ni, 98.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    440320k total,   140280k used,   300040k free,    28116k buffers
Swap:  1502036k total,        0k used,  1502036k free,    58160k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7170 root      20   0  2408 1108  880 R  0.7  0.3   0:00.06 top
 3092 root      20   0 87148 9460 1792 S  0.3  2.1   1:28.12 python
    1 root      20   0  2116  688  588 S  0.0  0.2   0:01.04 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      15  -5     0    0    0 S  0.0  0.0   0:00.84 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:01.24 watchdog/0
    6 root      15  -5     0    0    0 S  0.0  0.0   0:03.30 events/0
```

Figure 65: “Top” Command running in Debian Linux

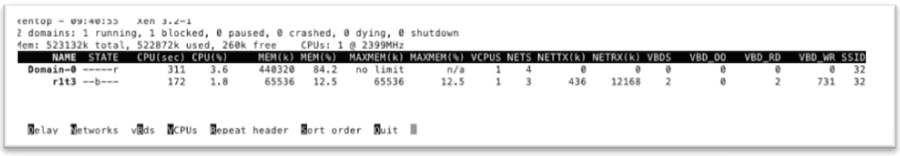
7.3 Monitoring of virtual resources based on SNMP and system tools

To analyze the capabilities of a virtual router the monitoring of the resources in the physical and the virtual machines are basic. The first step is to decide which parameters we will analyze to know the limitations in our environment.

There are two basic parameters that show how a physical or virtual machine consumes resources; the RAM consumption and the CPU Load. In our case, these parameters will show why virtual routers have limitations sending packets through a network. These parameters can be monitored directly in a Linux operative system using the “top” command existing in UNIX systems. This command shows tasks running, CPU Load and memory consumption for the system and for the commands running. Figure 65 shows a capture of “top” command in Debian Linux.

The “top” command is useful to see the parameters we would like, but it is necessary to stay logged in the system and under heavy load conditions we can’t see this parameters due to system fluctuations. If we assume that we must monitor the Physical machine where Xen Hypervisor is installed and the virtual machine that we will analyze it means that we must be logged in two systems that will be stressed by our tests.

An alternative to the “top” command is the “xentop” command. This command present in the Xen Hypervisor system, allows us to see the basic parameters of the Xen Hypervisor system (Dom0) and the virtual machines running (DomU’s) directly from the Dom0 system. It shows us the domain name, state of it, CPU Load in seconds and in percentage, memory



```
xentop - 09:40:55   Xen 3.2-1
2 domains: 1 running, 1 blocked, 0 paused, 0 crashed, 0 dying, 0 shutdown
Mem: 523132k total, 522872k used, 260k free    CPUs: 1 @ 2399MHz
      NAME  STATE   CPU(sec) CPU(%)     MEM(k) MEM(%)  MAXMEM(k) MAXMEM(%) VCPUS NETS NETTX(k) NETRX(k) VBDS    VBD_OO    VBD_RD    VBD_WR SSID
  Domain-0 -----r        311    3.6     440320   84.2   no limit       n/a     1    4        0        0    0        0         0         0   32
      r1t3 --b---        172    1.8      65536   12.5      65536      12.5     1    3      436    12168    2        0         2       731   32



  Delay  Networks  vBds  VCPUs  Repeat header  Sort order  Quit
```

Figure 66: “xentop” command output in Xen Hypervisor System

assigned and maximum memory that can be assigned in Kbytes and in percentage, virtual CPU’s, number of network interfaces and transmitted or received Kbytes, and also disks I/O.

This command will be useful in our monitoring system because we can monitor the Dom0 and the DomU at the same time without login in both operative system. Figure 66 shows the output of “xentop” command.

In Figure 66 we can see that there is a virtual machine named “r1t3” and also the Domain0 with their information explained previously. The problem of “xentop” is the same of “top” command; if we stress the system, the parameters are not shown steady.

The best alternative to monitor a system without login is SNMP. SNMP (Simple Network Management Protocol). Using this protocol we can access a lot of parameters of managed systems making queries to the agent installed on managed systems or directly receiving the stats automatically from the agent.

The information or variables accessible via SNMP protocol are organized in hierarchies described by MIBs (Management Information Bases). Using SNMP we will be able to use SNMP command-line applications to monitor a variable or change some variables to take control of an operative system without login. The requirements to monitor via SNMP are to install the SNMP command-line applications in the monitor system to make the request to the agents, and to install the SNMP devel (snmpd) in the agents to reply the requests.

Using SNMP queries directly to the virtual and physical systems, we can monitor RAM consumption, but the CPU queries don’t show reliable information of CPU Load. We realised that fact comparing the information of “top” and “xentop” with the information showed by the SNMP. To solve this problem we decided to create a script using perl that monitors every 5 seconds the information showed by the “xentop” command, and saves it to a file that will be accessible remotely via SNMP.

At this point we can monitor RAM consumption and CPU Load without login in the system and saving this information if a file to analyze it.

7.3.1 Virtual network scenario setup

To analyze the virtual routers capabilities we used a basic testbed running a virtual router and different network configurations.

Scenario Topology

Our testbed if formed by three physical machines and one virtual router. In Figure 67 we can see the physical and virtual testbed topology.

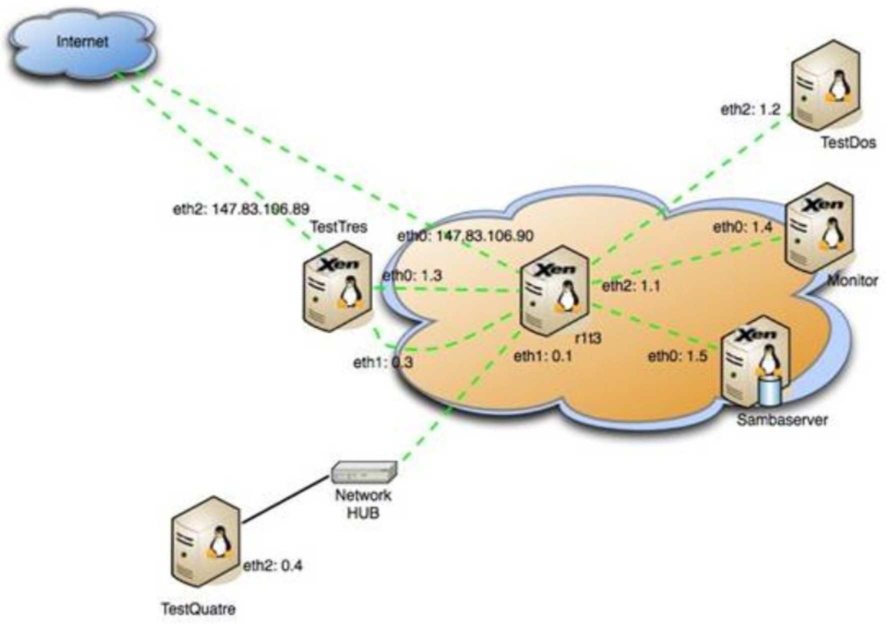


Figure 67: Physical and virtual testbed topology

In the picture we can see physical testbed formed by three machines, TestDos, TestQuatre and TestTres. TestQuatre and TestDos are connected to TestTres using Ethernet interfaces and TestTres is also connected to Internet. The virtual topology shows “r1t3” running over TestTres and virtually connected to TestDos, TestQuatre and Internet.

Physical and virtual machines configuration

TestDos and TestQuatre are running Debian and have “Iperf” tool installed to generate packet frames over the virtual router. The previous figure also shows that these physical machines are in different subnets to make sure that the virtual router has postrouting capabilities. TestTres is the machine under test. Is also running Debian Lenny with Xen Hypervisor installed that allows us to run virtual machines over the system. In this case we are running one virtual machine, “r1t3” with Debian Lenny and 64 MB of dedicated RAM.

Network Configurations

The Xen Hypervisor allows users to run different network configurations. It can share the physical network resources with the virtual machines using routing or bridged configurations, and also it can dedicate one or multiple network cards to the virtual resources. In this testbed we only used bridged and dedicated configurations because routing configuration has worse performance.

Bridged configuration

In the bridged configuration, the Domain0 or Xen Hypervisor is sharing the physical network card with the virtual machines. To control the Ethernet traffic and place it to the correct Domain, the Xen Hypervisor creates a bridge in the physical machine that is connected to the physical interface and to a virtual interface to share the traffic with the virtual machine. Figure 68 shows the bridged configuration in TestTres machine.

In the figure we can appreciate how works the Xen Hypervisor with bridged configuration.

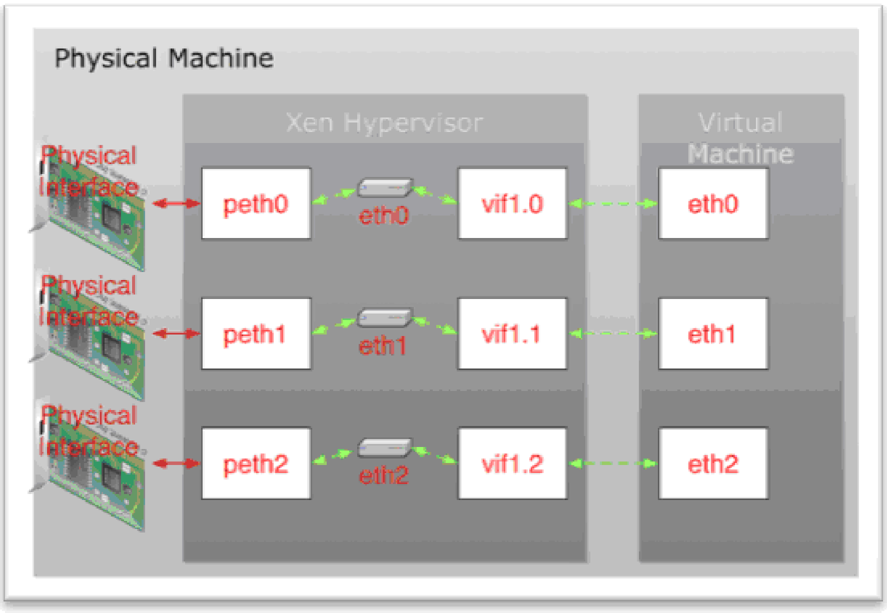


Figure 68: Bridged configuration

On system power on, it creates a bridge for any physical interface present in the machine. This bridge will be named with the name of physical interface. The Xen Hypervisor changes the physical interface name to “pethX”, where “X” is the number of the interface, and “peth” means physical Ethernet.

When a new virtual machine is created, Xen also creates one virtual interface in Dom0 for any interface defined in the virtual machine configuration file, and it will be connected to the bridge specified in this file. This virtual interface will be named “vifX.Y” where “X” is the number of virtual machine, “Y” is the number of interface in the virtual machine, and “vif” means virtual interface. We can see that our virtual router will have three interfaces connected to the three physical interfaces to communicate with TestQuatre, TestDos and Internet. In this case, “r1t3” can also communicate with TestTres because the bridges created in the physical machine shares the Ethernet connection. Now if we send a packet to the virtual machine, it will arrive by the physical interface of his subnet, and the bridge will read the direction and deliver to the corresponding virtual interface. For non-virtualized machines, this configuration is like they were directly connected to the virtual router without knowledge that there are two machines connected to the same physical interface.

Dedicated network interface configuration

The Xen Hypervisor also allows dedicating physical devices to the virtual machines. With this configuration the device is only visible by the virtual machine and the packets don’t pass through any bridge because the physical interface directly delivers the packets to the virtual machine. Figure 69 shows the dedicated network interface configuration.

In this case when the physical system is switched on, the interfaces are visible for the Domain0. Then we can unbind the interface from the system. In the virtual machine configuration file we can specify which interfaces will be dedicated to our virtual router and when we turn-on it, the interfaces will appear in the virtual machine. Now if a packet arrives to the physical interface, it directly pass to the controller in the virtual machine to read it.

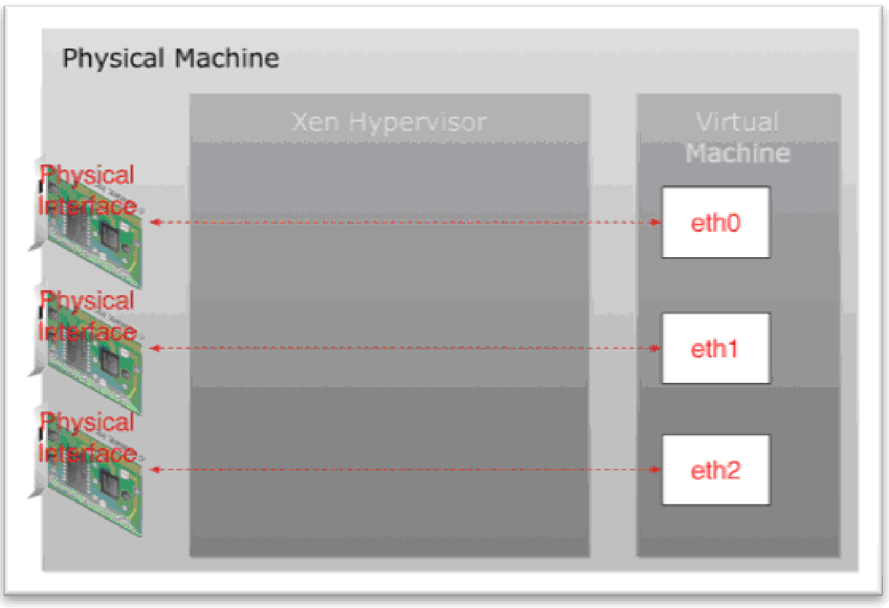


Figure 69: Bridged configuration

7.3.2 Test plan

We have tested our virtual router to observe his routing performance; in other words, we were interested to investigate its packet loss rate, the cause of packet losses and compare it with the performance of a physical router. To make our tests we use “Iperf” installed on TestDos and TestQuatre.

“Iperf” allows sending packets between two machines and allows us to change the packet length, the sending bandwidth and the time sending packets. With these three variables, we are able to send a given number of packets of the same length. “Iperf” also reports statistics of number of packets sent, number of packets lost, time to sent the packets and jitter. With these capabilities we decided to configure “Iperf” to make three different tests sending different number of packets of different length through the router. Specifically, we have created a perl script in TestQuatre with three basic functions:

1. Use Iperf to send different number of packets with different lengths
2. Make SNMP requests to Xen Hypervisor to consult the CPU Load and RAM consumption
3. Save the number of sent and lost packets to analyze the virtual router capabilities.

The packet length was set to 64, 256 and 1500 bytes, and the bandwidth augmented from 1Mbps to 101 Mbps in steps of 4 Mbps. With these tests applied to the different network configurations, the plan is to see the correlation between the number of packets transmitted per second through the router and the CPU load and RAM consumption of the physical and virtual machines. With this information we will analyze the capabilities of Xen to run an efficient virtual router.

7.3.3 Evaluation results

At this point we present the results of the tests in two types of graphs (Figures 70 through 82). One showing the evolution of receives packets versus transmitted ones (two curves each). The other type presents received packets (packets per second), CPU Loads (percentage of total) of the physical machine (PM CPU), the virtual router (VR CPU) and the total of physical machine (Total CPU). In all cases the X-Axis shows the number of packets sent by the traffic source TestQuatre.

Results for Bridged Configuration

Figures 70 through 73 reveal that the principal limitation with this configuration is the number of packets passing through the virtual router. In Figures 70 and 71 we can see that at the beginning the rate of received packets is similar to transmitted ones but at a given point, received packets stabilize around 16.000. This occurs when the CPU reaches 100% utilization. In Figure 72 we observe that received packets are similar to transmitted ones but keep in mind that the maximum rate in this case is around 8000 packets second due to the longer packet size used.

Analyzing the Figure 73, we can observe the effect of CPU saturation in received packets. We can see how the total CPU utilization grows at the same time of number of packets received. When CPU Load arrives at 100%, we can appreciate that packets received decrease in front of packets transmitted.

Results for dedicated network interface configuration

Figures 74, 76 and 78 reveal an optimal behavior in the sense that received packets follow well transmitted ones. We will see hereafter that this is like we had a direct connection between transmitter and receiver. Figures 75, 77 and 79 show the CPU Load. It is important to observe in this case that the virtual router basically use all the CPU Load.

Comparing the results obtained with these network configurations, we conclude that the principal problem using a virtual machine as a router is the control of the Ethernet interface. The domain that controls this interface takes up the CPU during most of the time. If this domain corresponds to the Xen Hypervisor, the virtual router can't access to the CPU to read the packets and therefore these ones become lost. In case the virtual router takes the control of network interfaces it can also dedicate the CPU time to deliver the packets to their destination.

Results for sender and receiver directly connected

Figures 80 through 82 show the comparison of transmitted and received packets when the machines running Iperf are directly connected using an Ethernet cable. These graphs show us the limitations of sender and receiver with constant length of packet.

We can appreciate that there's a limitation of number of packets transmitted by the sender (TestQuatre); this limitation is around 88.000 packets per second. Also we can appreciate a limitation in reception. The limitation of Testdos is around 83.000 packets per second.

Also we observe a limitation when the packet length grows. In fact, for packets of 256 bytes we can't send more than 39.000 packets per second and for lengths of 1500 bytes this limitation is around 8.000 packets per second. Also we can see that for this packet length there is no problem in reception. This information is helpful as a reference to evaluate the results using a virtual router.

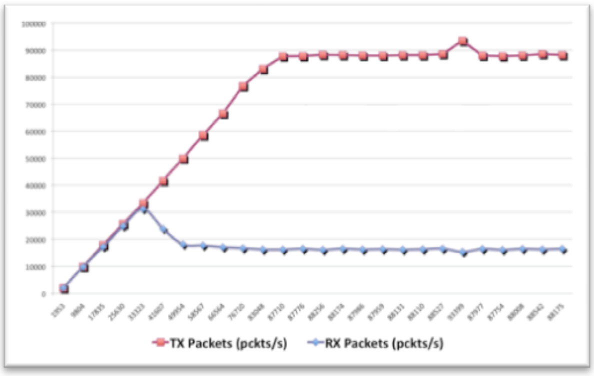


Figure 70: TX and RX packets 64 bytes length. Bridged configuration

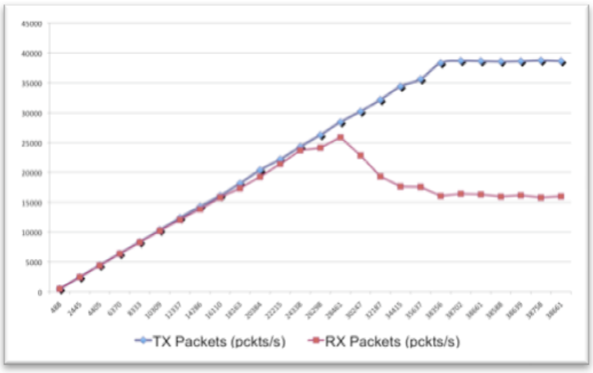


Figure 71: TX and RX packets 256 bytes length. Bridged configuration

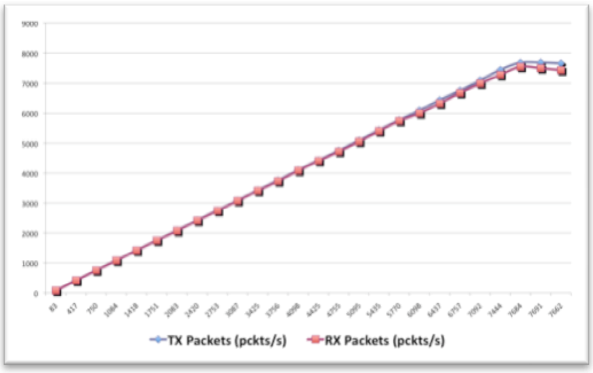


Figure 72: TX and RX packets 1500 bytes length. Bridged configuration

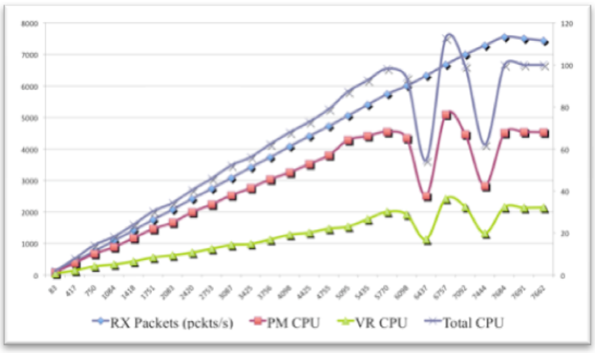


Figure 73: RX packets 1500 bytes length and CPU Load. Bridged configuration

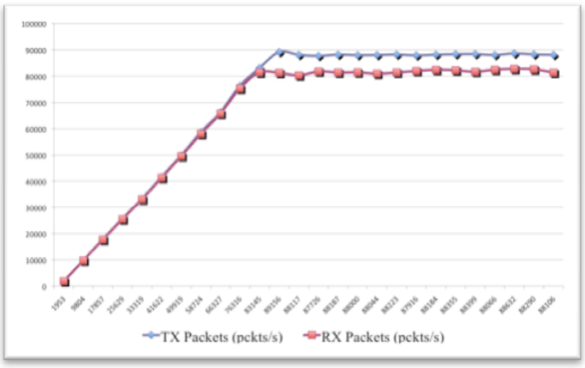


Figure 74: TX and RX packets 64 bytes length. Dedicated interface

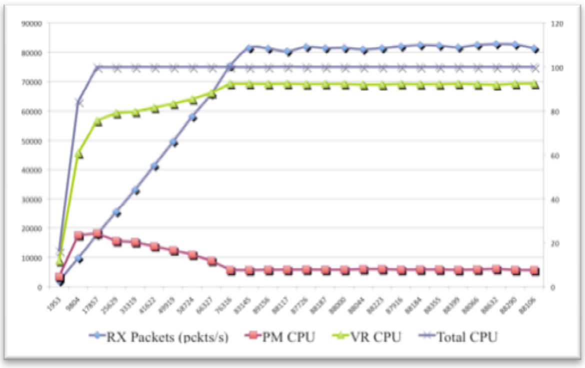


Figure 75: RX packets 64 bytes length and CPU Load. Dedicated interface

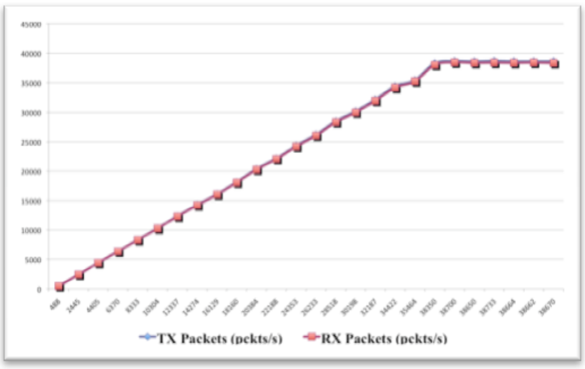


Figure 76: TX and RX packets 256 bytes length. Dedicated interface

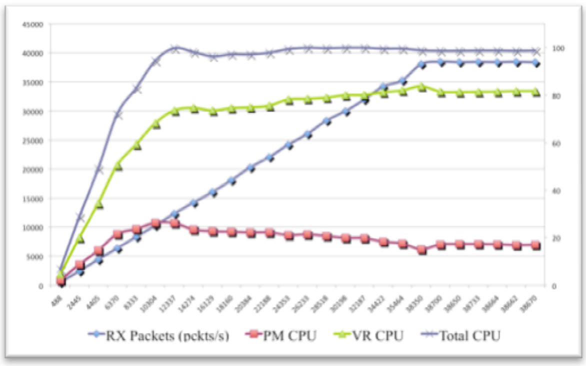


Figure 77: RX packets 256 bytes length and CPU Load. Dedicated interface

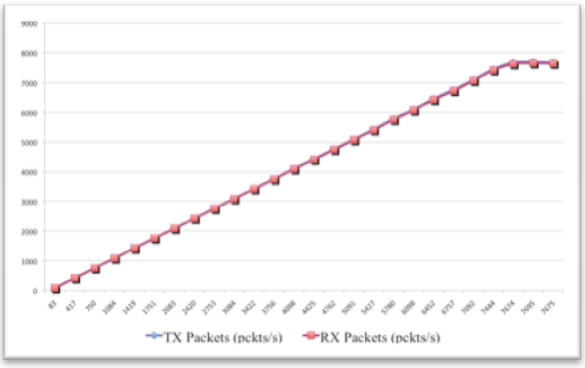


Figure 78: TX and RX packets 1500 bytes length. Dedicated interface

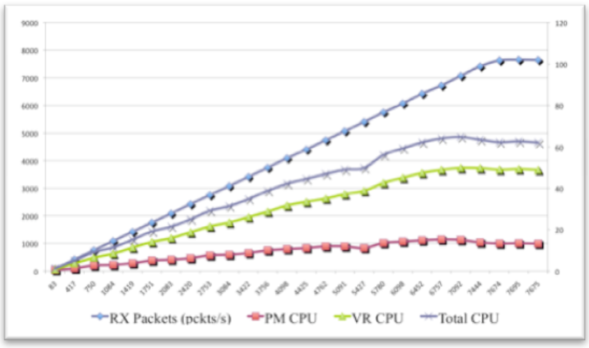


Figure 79: Rx packets 1500 bytes length and CPU Load. Dedicated interface

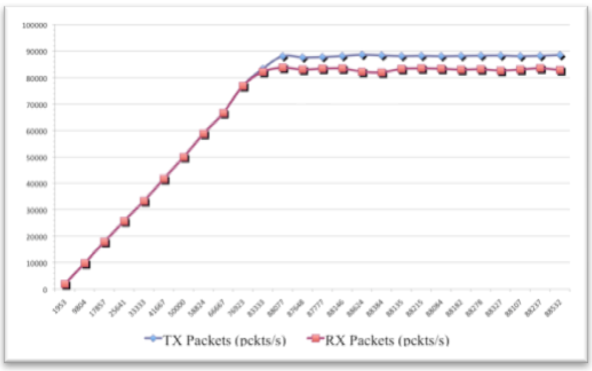


Figure 80: TX and RX packets 64 bytes length. Direct connection

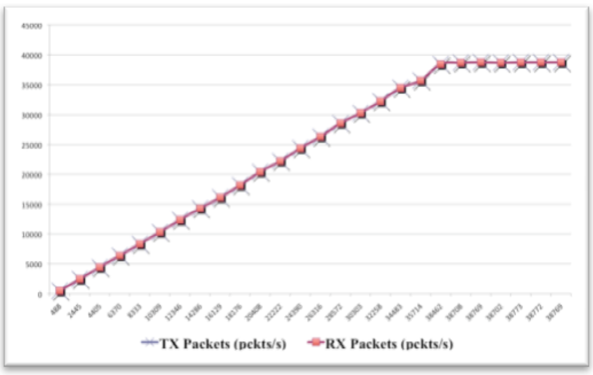


Figure 81: TX and RX packets 256 bytes length. Direct connection

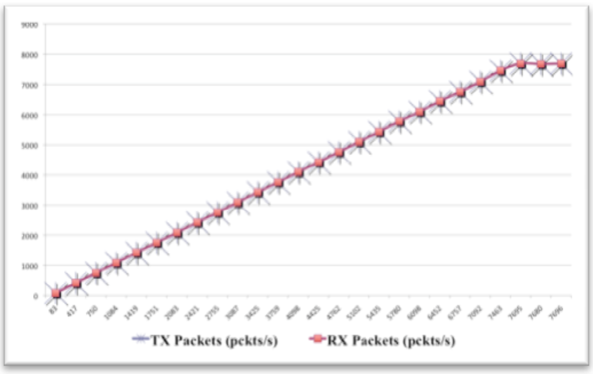


Figure 82: TX and RX packets 1500 bytes length. Direct connection

7.4 Monitoring of virtual resources in EmanicsLab

Elabmoni is one of the monitoring applications used for EmanicsLab. Its web-based interface displays information about the resource utilization of the slices which are running in the EmanicsLab testbed. This information is also stored in a MySQL database. The original version of the application was running in the host environment. In order to improve security, Elabmoni was modified in order to be able to run in a regular EmanicsLab slice. The VSys tool is used in order to allow the Elabmoni application to gather information about the resource utilization of other slices running on the same node. VSys allows scripts, running on the host machine, to be executed in a controlled manner from the virtual machines.

7.4.1 Elabmoni Architecture

Figure 83 illustrates the Elabmoni architecture. The clients are running on regular slices in different EmanicsLab nodes. As mentioned, they use the VSys tool in order to execute a script that gathers information about the resource utilization of slices running in the same nodes. This information is sent to the Elabmoni collector, which stores it in a MySQL database. Figure 84 presents the schema of the table in which this statistical information is stored.

The web-based GUI was developed using PHP. It accesses the MySQL database and displays the statistical information related to the resource utilization of different slices and nodes. Figure 85 shows the Elabmoni GUI.

7.5 Conclusions

Effort in VirtMon project has been devoted to propose and analyze a monitoring architecture for virtual devices, to investigate the consumption of computational resources by different types of virtual router configurations and to implement the monitoring mechanism in EmanicsLab. The three have in common the utilization of more basic tools to carry out the monitoring process. In fact, the proposed monitoring architecture of section 7.2 makes use of libvirt, the monitoring of computational resources described in section 7.3 is based

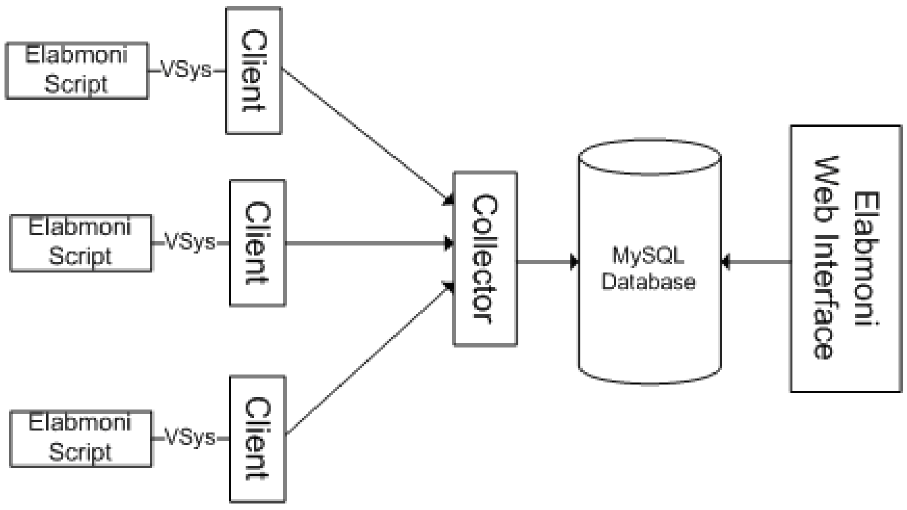
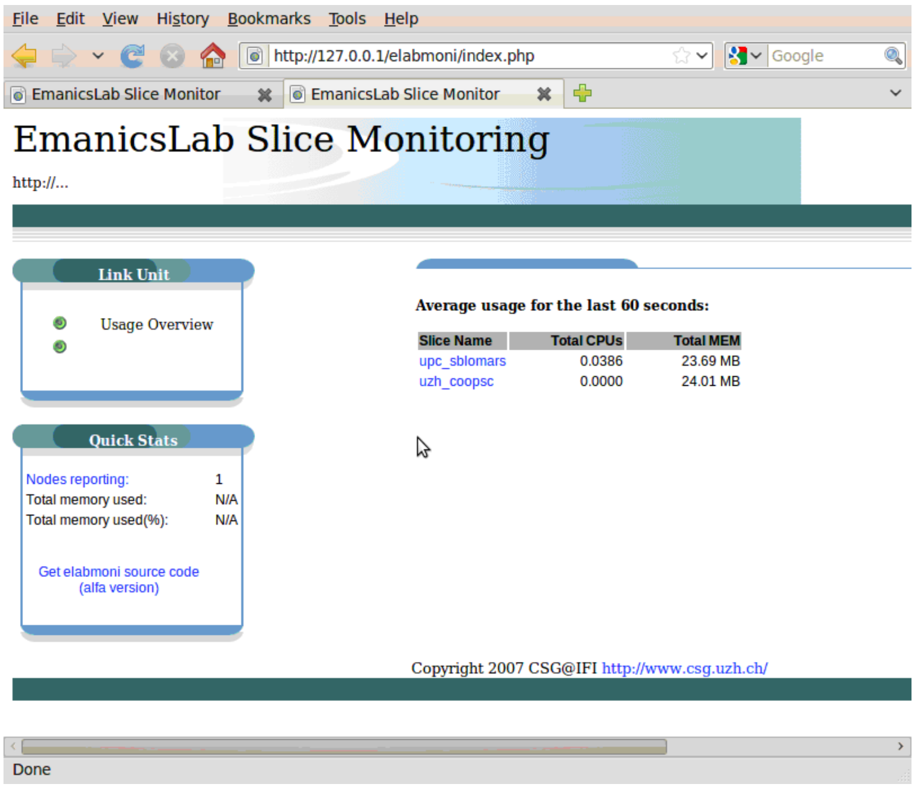


Figure 83: Elabmoni Architecture

```
CREATE TABLE ‘slice-usage‘ (
‘timestamp‘ int(11) NOT NULL,
‘hostname‘ varchar(255) NOT NULL,
‘slicename‘ varchar(255) NOT NULL,
‘context‘ int(11) NOT NULL,
‘cpu‘ float NOT NULL,
‘mem‘ float NOT NULL,
‘rss‘ int(11) NOT NULL,
```

Figure 84: MySQL Schema



EmanicsLab Slice Monitoring

http://...

Average usage for the last 60 seconds:

Slice Name	Total CPUs	Total MEM
upc_sblomars	0.0386	23.69 MB
uzh_coopsc	0.0000	24.01 MB

Nodes reporting:	1
Total memory used:	N/A
Total memory used(%):	N/A

Get elabmoni source code (alfa version)

Copyright 2007 CSG@IFI http://www.csg.uzh.ch/

Figure 85: Elabmoni Screenshot

on Xen utilities and SNMP, whereas the EmanicsLab virtual monitoring tool is based on VSys. All these solutions proved to be effective. Moreover, the architectural approach has been used in the context of an information management overlay platform, which monitors network and system parameters at real-time and regulates information flow based on the properties and state of the network environment.

Particular attention has to be paid to the research of consumption of resources by virtual routers according to the configuration adopted. We have seen that using bridged configuration the number of packets per second that can pass through our virtual router are around 20% of the total packets transmitted. This is because any packet arriving to the interface has to be read and if his destination is the virtual router it has to be put on the virtual interface. This operation will repeat when the packet pass the router and goes to his destination. The above causes an increment of CPU Load and limit the virtual router performance. Nevertheless, using the dedicated network interface configuration, we have a better performance and practically there isn't any packet loss. We have a better performance because there is only one system that must read the packets and it works like a physical machine. The problem with this configuration is in case we need to create more than one virtual machine (virtual router). Then, the domains not associated to a given interface can't use it with the corresponding lack of connectivity.

We can confirm that a virtual router is feasible using the Xen Hypervisor and this router will have the same capabilities of any Linux router, but we must use more powerful machines if we require a good system performance.

8 Conclusions

We have reported in this deliverable on the scientific work performed in work package 7 in 2009. This work has been structured around the topics of security, monitoring, and configuration in large scale environments. An open call for activity proposals was initiated at the end of 2008 and finalized at the beginning of 2009. Five activities that cover different aspects of the above mentioned topics were chosen to be funded during the last phase of the EMANICS project, namely

- Extended Scalable Management of Biometric devices (BioScale II),
- Flow-based Monitoring and Anomaly Detection (FMAD),
- Management of large MANETs (MaMANET),
- Security Management Infrastructure (SeMI),
- Virtualization Monitoring (VirtMon).

In this deliverable, we have described the goals of those five activities as well as the results achieved by them in the reporting period. The reported contents shows that the activities performed very well, resulting in several peer-reviewed high-quality publications at international conferences and journals. In addition, in 2009 the members of the work package were involved in the organization of several events, such as the EMANICS Workshop on Netflow Usage, allowing an effective exchange of knowledge and results between the EMANICS partners and other international experts.

9 Acknowledgment

This deliverable was made possible due to the large and open help of the WP7 Partners of the EMANICS NoE. Many thanks to all of them.

References

[1] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A Pras, and B. Stiller. An overview of ip flow-based intrusion detection. 2010 (To appear).

[2] A. Sperotto, G. Vliek, R. Sadre, and A. Pras. Detecting spam at the network level. 2009.

[3] A. Sperotto, R. Sadre, P. T. de Boer, and A. Pras. Hidden markov model modeling of ssh brute-force attacks. In *Integrated Management of Systems, Services, Processes and People in IT, Proceedings of the 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2009, Venice, Italy*, volume 5841/2009 of *Lecture Notes in Computer Science*, pages 164–176. Springer Verlag, October 2009.

[4] Computer Economics. 2007 malware report: The economic impact of viruses, spyware, adware, botnets, and other malicious code, Jul. 2008.

[5] M. Roesch. Snort, intrusion detection system, Jul. 2008.

[6] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[7] H. Lai, S. Cai, H. Huang, J. Xie, and H. Li. A parallel intrusion detection system for high-speed networks. In *Proc. of the Second International Conference Applied Cryptography and Network Security (ACNS'04)*, pages 439–451, May 2004.

[8] M. Gao, K. Zhang, and J. Lu. Efficient packet matching for gigabit network intrusion detection using TCAMs. In *Proc. of 20th International Conferece on Advanced Information Networking and Applications (AINA'06)*, pages 249–254, 2006.

[9] L.T. Heberlein, G.V. Dias, K.N. Levitt, B. Mukherjee, J.Wood, and D. Wolber. A network security monitor. In *Proc. of IEEE Computer Society Symposium on Research in Security and Privacy*, pages 296–304, May 1990.

[10] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagl, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS - a graph based intrusion detection system for large networks. In *Proc. of the 19th National Information Systems Security Conference (NISS '96)*, pages 361–370, 1996.

[11] Cisco.com. Cisco IOS NetFlow Configuration Guide, Release 12.4. http://www.cisco.com, Jul. 2008.

[12] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), Jul. 2008.

[13] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917 (Informational), Jul. 2008.

[14] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(9):805–822, Apr. 1999.

[15] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion detection systems. *Annales des Telecommunications*, 55(7–8):361–378, Jul. 2000.

[16] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., March 2000.

[17] Hervé Debar and Jouni Viinikka. Intrusion detection: Introduction to intrusion detection and security information management. In *Foundations of Security Analysis and Design III*, pages 207–236, Sep. 2005.

[18] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. Intrusion detection: A survey. *Managing Cyber Threats*, pages 19–78, June 2005.

[19] CERT Coordination Center. http://www.cert.org/certcc.html, Jul. 2008.

[20] Internet2 NetFlow Weekly Reports. http://netflow.internet2.edu/weekly, Jul. 2008.

[21] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proc. SIGSAC: 11th ACM Conference on Computer and Communications Security (CSS'04)*, pages 2–11, 2004.

[22] Z.M. Fadlullah, T. Taleb, N. Ansari, K. Hashimoto, Y. Y. Miyake, Y. Nemoto, and N.Kato. Combating against attacks on encrypted protocols. In *IEEE International Conference on Communications (ICC '07).*, pages 1211–1216, June 2007.

[23] T. Taleb, Z. Md. Fadlullah, K. Hashimoto, Y. Nemoto, and N. Kato. Tracing back attacks against encrypted protocols. In *Proc. of the 2007 international conference on Wireless communications and mobile computing (IWCMC '07)*, pages 121–126, 2007.

[24] S. Song and Z. Chen. Adaptive network flow clustering. In *IEEE International Conference on Networking, Sensing and Control (ICNSC07)*, pages 596–601, April 2007.

[25] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), Jul. 2008.

[26] Gregor Schaffrath and Burkhard Stiller. Conceptual integration of flow-based and packet-based network intrusion detection. In *Proc. of 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS '08)*, pages 190–194, 2008.

[27] Tiago Fioreze, Mattijs Oude Wolbers, Remco van de Meent, and Aiko Pras. Finding elephant flows for optical networks. In *Proc. of 10th IFIP/IEEE International Symposium on Integrated Network Management (IM '07)*, pages 627–640, May 2007.

[28] S. Leinen. Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX). RFC 3955 (Informational), Jul 2008.

[29] Packet Sampling (PSAMP) working group. http://www.ietf.org/html.charters/psamp-charter.html, Jul. 2008.

[30] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *Proc. of the 6th ACM SIGCOMM conference on Internet measurement (IMC '06)*, pages 159–164, 2006.

[31] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is sampled data sufficient for anomaly detection? In *Proc. of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC'06)*, pages 165–176, 2006.

[32] Tanja Zseby, Thomas Hirsch, and Benoit Claise. Packet sampling for flow accounting: Challenges and limitations. In *Proc. of 9th International Conference on Passive and Active Measurement (PAM'08)*, pages 61–71, 2008.

[33] E. Izkue and E. Magaña. Sampling time-dependent parameters in high-speed network monitoring. In *Proc. of the ACM international workshop on Performance monitoring, measurement, and evaluation of heterogeneous wireless and wired networks (PM2HW2N '06)*, pages 13–17, 2006.

[34] H. Wang, Y. Lin, Y. Jin, and S. Cheng. Easily-implemented adaptive packet sampling for high speed networks flow measurement. In *Proc. of 6th International Conference on Computational Science (ICCS'06)*, pages 128–135, 2006.

[35] G. He and J. C. Hou. An in-depth, analytical study of sampling techniques for self-similar internet traffic. In *Proc.of 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 404–413, 2005.

[36] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[37] Nick Duffield, Carsten Lund, and Mikkel Thorup. Flow sampling under hard resource constraints. *SIGMETRICS Perform. Eval. Rev.*, 32(1):85–96, 2004.

[38] N. Duffield, C. Lund, and M. Thorup. Learn more, sample less: control of volume and variance in network measurement. *IEEE Transactions on Information Theory*, 51(5):1756–1775, May 2005.

[39] Noga Alon, Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating arbitrary subset sums with few probes. In *Proc. of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '05)*, pages 317–325, 2005.

[40] V. Igure and R. Williams. Taxonomies of attacks and vulnerabilities in computer systems. *Communications Surveys & Tutorials, IEEE*, 10(1):6–19, 2008.

[41] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *Proc. of 2003 ACM workshop on Rapid malcode (WORM'03)*, pages 11–18, 2003.

[42] S. Hansman and R. Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, Feb. 2005.

[43] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security & Privacy*, 1(4):33–39, Jul.-Aug. 2003.

[44] J. D. Howard. *An analysis of security incidents on the Internet 1989-1995*. PhD thesis, Carnegie Mellon University, 1998.

[45] W. Lee, C. Wang, and D. Dagon. *Botnet Detection. Countering the Largest Security Threat*, volume 36. Spinger, 2008.

[46] David Dagon, Guofei Gu, and Christopher Lee. A taxonomy of botnet structures. In *Botnet Detection*, volume 36, pages 143–164, Oct. 2007.

[47] W. Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In Wenke Lee, Cliff Wang, and David Dagon, editors, *Botnet Detection*, volume 36, pages 1–24, 2008.

[48] L. R. Halme and R. K. Bauer. AINT misbehaving – A taxonomy of anti-intrusion techniques. In *Proc. of 18th NIST-NCSC National Information Systems Security Conference*, pages 163–172, 1995.

[49] Benjamin Morin and Ludovic Mé. Intrusion detection and virology: an analysis of differences, similarities and complementariness. *Journal in Computer Virology*, 3:39–49, Apr. 2007.

[50] P. Li, M. Salour, and X. Su. A survey of internet worm detection and containment. *IEEE Communications Surveys & Tutorials*, 10(1):20–35, 2008.

[51] Moses Garuba, Chunmei Liu, and Duane Fraites. Intrusion techniques: Comparative study of network intrusion detection systems. In *Proc. of 5th International Conference on Information Technology: New Generations (ITNG '08)*, pages 592–598, Apr. 2008.

[52] M. Almgren, E. L. Barse, and E. Jonsson. Consolidation and evaluation of IDS taxonomies. In *Proc. of 8th Nordic Workshop on Secure IT systems (NordSec '03)*, Oct. 2003.

[53] D. Moore, C. Shannon, D. J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet denial-of-service activity. *ACM Transactions on Computer Systems*, 24(2):115–139, May 2006.

[54] Z. Li, Y. Gao, and Y. Chen. Towards a high-speed router-based anomaly/intrusion detection system. http://conferences.sigcomm.org/sigcomm/2005/poster-121.pdf, Aug. 2005.

[55] Y. Gao, Z. Li, and Y. Chen. A dos resilient flow-level intrusion detection approach for high-speed networks. In *Proc. of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS '06)*, page 39, 2006.

[56] S. M. Specht and R. B. Lee. Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *Proc. of the ISCA 17th International Conference on Parallel and Distributed Computing Systems (ISCA PDCS'04)*, pages 543–550, Sep. 2004.

[57] Q. Zhao, J. Xu, and A. Kumar. Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation. *IEEE Journal on Selected Areas in Communications*, 24(10):1840–1852, Oct. 2006.

[58] M.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J.W. Hong. A flow-based method for abnormal network traffic detection. In *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS'04)*, pages 599–612, Apr. 2004.

[59] G. Münz and G. Carle. Real-time analysis of flow data for network attack detection. In *Proc. of 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, pages 100–108, 2007.

[60] Diadem Firewall European Project. http://www.diadem-firewall.org, Jul. 2008.

[61] C. Diot A. Lakhina, M. Crovella. Characterization of network-wide anomalies in traffic flows. In *Proc. of 4th ACM SIGCOMM conference on Internet measurement (IMC '04)*, pages 201–206, 2004.

[62] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228, 2005.

[63] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. *SIGMETRICS Perform. Eval. Rev.*, 32(1):61–72, 2004.

[64] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proc. of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '04)*, number 4, pages 219–230, 2004.

[65] Sprint.net. http://www.sprint.net, Jul. 2008.

[66] A. Sperotto, R. Sadre, and A. Pras. Anomaly characterization in flow-based traffic time series. In *Proc. of the 8th IEEE International Workshop on IP Operations and Management, IPOM 2008, Samos, Greece*, pages 15–27, Sep. 2008.

[67] SURFnet. www.surfnet.nl, Jul. 2008.

[68] Arno Wagner and Bernhard Plattner. Entropy based worm and anomaly detection in fast IP networks. In *Proc. of 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE '05)*, pages 172–177, June 2005.

[69] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[70] C. Gates, J.J. McNutt, J.B. Kadane, and M.I. Kellner. Scan detection on very large networks using logistic regression modeling. In *Proc. of 11th IEEE Symposium on Computers and Communications (ISCC'06)*, pages 402–408, 2006.

[71] Marc Stoecklin, Jean-Yves Le Boudec, and Andreas Kind. A two-layered anomaly detection technique based on multi-modal flow behavior models. In *Proc. of 9th International Conference on Passive and Active Measurement (PAM'08)*, pages 212–221, 2008.

[72] Thomas Dübendorfer and Bernhard Plattner. Host behaviour based early detection of worm outbreaks in internet backbones. In *Proc. of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, pages 166–171, 2005.

[73] Thomas Dübendorfer, Arno Wagner, and Bernhard Plattner. A framework for real-time worm attack detection and backbone monitoring. In *Proc.of 1st IEEE International Workshop on Critical Infrastructure Protection (IWCIP' 05)*, Nov. 2005.

[74] A.Wagner, T. Dübendorfer, B. Plattner, and R. Hiestand. Experiences with worm propagation simulations. In *Proc. of 2003 ACM workshop on Rapid malcode (WORM'03)*, pages 34–41, 2003.

[75] Minsoo Lee, Taeshik Shon, Kyuhyung Cho, Manhyun Chung, Jungtaek Seo, and Jongsub Moon. An approach for classifying internet worms based on temporal behaviors and packet flows. In *Proc. of 3rd Int. Conf. on Intelligent Computing (ICIC 2007)*, pages 646–655, 2007.

[76] C.C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *Proc. of 17th USENIX Security Symposium (USENIX Security '08)*, pages 138–147, 2002.

[77] F. Dressler, W. Jaegers, and R. German. Flow-based worm detection using correlated honeypot logs. In *Proc. of 15th GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007)*, pages 181–186, Feb. 2007.

[78] M. Collins and M. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *Proc. of 10th International Symposium on Recent Advances in Intrusion Detection (RAID'07)*, pages 276–295, 2007.

[79] D. Hoeflin A. Karasaridis, B. Rexroad. Wide-scale botnet detection and characterization. In *Proc.of the first conference on First Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, pages 1–8, 2007.

[80] C. Livadas, R. Walsh, D. Lapsley, and W.T. Strayer. Using machine learning techniques to identify botnet traffic. In *Proc. of 31st IEEE Conference on Local Computer Networks (LCN'06)*, pages 967–974, 2006.

[81] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. of 17th USENIX Security Symposium (USENIX Security '08)*, pages 139–154, June 2008.

[82] Z. Zhu, G. Lu, Y. Chen, Z.J. Fu, P. Roberts, and K. Han. Botnet research survey. In *32nd Annual IEEE International Computer Software and Applications (COMPSAC '08)*, pages 967–972, Aug. 2008.

[83] Q. Xiaofeng, H. Jihong, and C. Ming. Flow-based anti-spam. In *Proc. of 4th IEEE Workshop on IP Operations and Management (IPOM'04)*, pages 99–103, Oct. 2004.

[84] Symantec.com. The state of spam, a monthly report - july 2008. http://www.symantec.com/, Jul. 2008.

[85] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. *SIGCOMM Comput. Commun. Rev.*, 36(4):291–302, 2006.

[86] Spamassassin. http://spamassassin.apache.org, March 2009.

[87] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A Pras, and B. Stiller. An Overview of IP Flow-based Intrusion Detection. *To appear: IEEE Surverys & Tutorials*, 2009.

[88] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proc. of the 14th ACM conference on Computer and Communications Security (CCS '07)*, 2007.

[89] D. Schatzmann, M. Burkhart, and T. Spyropoulos. Flow-level Characteristics of Spam and Ham. Technical Report TIK Report Nr. 291, Computer Engineering and Networks Laboratory, ETH, Zurich, August 2008.

[90] D. Schatzmann, M. Burkhart, and T. Spyropoulos. Inferring Spammers in the Network Core. In *Proc. of 10th International Conference on Passive and Active Network Measurement (PAM '09)*, 2009.

[91] P. Desikan and J. Srivastava. Analyzing Network Traffic to Detect E–Mail Spamming Machines. In *Proc. of the 2004 ICDM Workshop on Privacy and Security Aspects of Data Mining (PSDM '04)*, 2004.

[92] B.-C. Cheng, M.-J. Chen, Y.-S. Chu, A. Chen, S. Yap, and K.-P. Fan. SIPS: A Stateful and Flow-Based Intrusion Prevention System for Email Applications. In *Proc. of IFIP International Conference on Network and Parallel Computing (NPC '07)*, 2007.

[93] M. Žádník and Z. Michlovský. Is spam visible in flow-level statistic? Technical report, CESNET technical report 6/2008, 2008.

[94] A. Iverson. Blacklist statistic center. http://stats.dnsbl.com/, March 2009.

[95] A. Sperotto and R. van de Meent. A survey of the high-speed self-learning intrusion detection research area. In *First International Conference on Autonomous Infrastructure, Management and Security (AIMS 07)*, June 2007.

[96] NfSen - Netflow Sensor. http://nfsen.sourceforge.net, May 2009.

[97] A. Sperotto, R. Sadre, F. van Vliet, and A. Pras. A labeled data set for flow-based intrusion detection. In *Proc. of the 9th IEEE International Workshop on IP Operations and Management (IPOM 2009)*, 2009.

[98] D. Brauckhoff, A. Wagner, and M. Mays. Flame: a flow-level anomaly modeling engine. In *Proc. of the conference on Cyber security experimentation and test (CSET'08)*, 2008.

[99] J. Sommers, V. Yegneswaran, and P. Barford. A framework for malicious workload generation. In *Proc. of the 4th ACM SIGCOMM conference on Internet measurement (IMC '04)*, 2004.

[100] F. Camastra and A. Vinciarelli. *Markovian Models for Sequential Data*. 2008.

[101] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41, 1970.

[102] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[103] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 1989.

[104] G. A. Fink. *Markov Models for Pattern Recognition: From Theory to Applications*. Springer-Verlag New York, Inc., 2007.

[105] C. V. Wright, F. Monrose, and G. M. Masson. HMM profiles for network traffic classification. In *Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC 2004)*, 2004.

[106] A. Dainotti, A. Pescapé, P. S. Rossi, F. Palmieri, and G. Ventre. Internet traffic modeling by means of hidden markov models. *Computer Networks*, 52(14), 2008.

[107] A. Dainotti, W. de Donato, A. Pescape, and P.S. Rossi. Classification of network traffic via packet-level hidden markov models. 2008.

[108] D. Gao, M. K. Reiter, and D. X. Song. Behavioral distance measurement using Hidden Markov Models. In *Proc. of Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006*, 2006.

[109] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. 1999.

[110] R. Khanna and H. Liu. System approach to intrusion detection using hidden markov model. In *Proceedings of the 2006 International Conference on Wireless communications and mobile computing (IWCMC '06)*, 2006.

[111] C. Seifert. Analyzing malicious ssh login attempts, September 2006.

[112] SANS Institute. Top-20 2007 Security Risks (2007 Annual Update). www.sans.org, May 2009.

[113] C. Andrieu and A. Doucet. Simulated annealing for maximum a posteriori parameter estimation of hidden markov models. *IEEE Transactions on Information Theory*, 46, 2000.

[114] S. Ubik and P. ejdl. Experience with passive monitoring deployment in geant2 network. `http://tnc2009.terena.org/schedule/presentations/show.php?pres_id=80`. Presented at TERENA Networking Conference 2009.

[115] Distributed monitoring application programmable interface. `http://mapi.uninett.no/`.

[116] D. Antoniades, A. Oslebo, and S. Ubik. Abw-short-timescale passive bandwidth monitoring. In *International Conference on Networking (ICN 2007), Martinique*, pages 49–49. IEEE Computer Society Press, April 2007.

[117] V. Marinov and J. Schönwälder. Design of an IP Flow Record Query Language. In *Proceedings of AIMS '08*, pages 205–210, Berlin, Heidelberg, 2008. Springer-Verlag.

[118] M. Sullivan and A. Heybey. Tribeca: a System for Managing Large Databases of Network Traffic. In *Proceedings of ATEC'98*, pages 13–24, Berkeley, CA, USA, 1998. USENIX Association.

[119] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in Data Stream Systems. In *Proceedings of PODS '02*, pages 1–16, New York, NY, USA, 2002. ACM.

[120] J.-P. Navarro, B. Nickless, and L. Winkler. Combining Cisco NetFlow Exports with Relational Database Technology for Usage Statistics, Intrusion Detection, and Network Forensics. In *Proceedings of LISA '00*, pages 285–290, Berkeley, CA, USA, December 2000. USENIX Association.

[121] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *Proceedings of SIGMOD'03*, pages 647–651, New York, NY, USA, June 2003. ACM.

[122] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of USENIX'93*, pages 259–270, Berkeley, CA, USA, 1993. USENIX Association.

[123] P. Haag. nfdump. http://nfdump.sourceforge.net/.

[124] D. Moore, K. Keys, R. Koga, E. Lagache, and KC. Claffy. The Coral Reef Software Suite as a Tool for System and Network Administration. In *Proceedings of LISA'01*, pages 133–144, Berkeley, CA, 2001. USENIX Association.

[125] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and KC. Claffy. The Architecture of CoralReef: an Internet Traffic Monitoring Software Suite. In *Proceedings of PAM'01*. CAIDA, RIPE NCC, 2001.

[126] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer. Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic. In *Proceedings of IMC'05*, Berkeley, CA, USA, 2005. USENIX Association.

[127] M. Fullmer. flow-tools. http://www.splintered.net/sw/flow-tools/.

[128] D. Plonka. FlowScan: A Network Traffic Flow Reporting and Visualization Tool. In *Proceedings of LISA '00*, pages 305–318, Berkeley, CA, USA, 2000. USENIX Association.

[129] T. Oetiker. RRDTool. http://oss.oetiker.ch/rrdtool/.

[130] C. Estan, S. Savage, and G. Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *Proceedings of SIGCOMM '03*, pages 137–148, New York, NY, USA, 2003. ACM.

[131] Michael Collins, Andrew Kompanek, and Timothy Shimeall. *Analysts' Handbook: Using SiLK for Network Traffic Analysis*. CERT, 0.10.3 edition, November 2006.

[132] V. Marinov. Design of an IP Flow Record Query Language. Master's thesis, Jacobs University Bremen, May 2009.

[133] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer. Information Model for IP Flow Information Export. RFC 5102, Cisco Systems, January 2008.

[134] A. Fin. A Genetic Approach to Qualitative Temporal Reasoning with Constraints. In *Proceedings of ICCIMA '99*, Washington, DC, USA, 1999. IEEE Computer Society.

[135] Symantec. *W32.Welchia.Worm*, August 2003.

[136] T. Dübendorfer, A. Wagner, T. Hossmann, and B. Plattner. Flow-level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone. In *Proceedings of DIMVA'05*, Vienna, Austria, July 2005. Springer's Lecture Notes in Computer Science (LNCS 3548).

[137] Fraunhofer FIRST. ReMIND ? real-time machine learning intrusion detection. `http://www.plixer.com`.

[138] James Moscola, Young H. Cho, and John W. Lockwood. Implementation of network application layer parser for multiple tcp/ip flows in reconfigurable devices. In *FPL*, pages 1–4. IEEE, 2006.

[139] Mayukh Dass, James Cannady, and Walter D. Potter. A blackboard-based learning intrusion detection system: a new approach. In *IEA/AIE'2003: Proceedings of the 16th international conference on Developments in applied artificial intelligence*, pages 385–390. Springer Springer Verlag Inc, 2003.

[140] A. Berg and L. Spaanenburg. Modular and Hierarchical Specialization in Neural Networks. *Berg, A., Spaanenburg, L., , Journal*, 3(1), 2003.

[141] Liu Mixia, Yu Dongmei, Zhang Qiuyu, and Zhu Honglei. Network security risk assessment and situation analysis. In *Proceedings of Passive and Active Measurement Workshop (PAM 2002)*, 2002.

[142] Huiqiang Wang, Ying Liang, and Haizhi Ye. An extraction method of situational factors for network security situational awareness. In *ICICSE '08: Proceedings of the 2008 International Conference on Internet Computing in Science and Engineering*, pages 317–320, Washington, DC, USA, 2008. IEEE Computer Society.

[143] S. Chakchai. A Survey of Network Traffic Monitoring and Analysis Tools. `http://www.cs.wustl.edu/~cs5/567/traffic/`.

[144] B. Reese. NetFlow versus sFlow. `http://www.techworld.com/networking/features/index.cfm?featureid=3865`, December 2007.

[145] NetQoS. What is Cisco ISO NetFlow? `http://www.netqos.com/resourceroom/initiatives/reporter-cisco-netflow.asp`.

[146] Hp procurve networking - application notes. `http://h40060.www4.hp.com/procurve/includes/application-notes/index.php?cc=uk&lc=en&content=ans6-en`.

[147] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176, InMon Corp., September 2001.

[148] J. Reeves and S. Panchen. Traffic monitoring with packet-based sampling for defense against security threats. In *Proceedings of the 2007 IEEE International Workshop on Anti-counterfeiting, Security, Identification*, 2007.

[149] Lancope. StealthWatch Architecture. `http://www.lancope.com/products/architecture.aspx`.

[150] Riverbed cascade, `http://www.riverbed.com/cascade/products/riverbed-nba.php`.

[151] Sourcefire rna, `http://www.sourcefire.com/products/3D/rna`.

[152] euromicron systems GmbH. Stealth watch. `http://www.lan-technik.de/produkte\_anomaly-detection\_lanco.98.html`.

[153] AKMA Labs Xharru Ltd. FlowMatrix Network Behavior Analysis System. `http://www.akmalabs.com/flowmatrix.php`.

[154] Claudio Moraga. Design of neural networks. In Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES (1)*, volume 4692 of *Lecture Notes in Computer Science*, pages 26–33. Springer, 2007.

[155] plixer International. Scrutinizer v6. `http://www.first.fraunhofer.de/en/remind`.

[156] Nemesis, command-line network packet crafting and injection utility, `http://nemesis.sourceforge.net`.

[157] gnuplot, `http://www.gnuplot.info`.

[158] Manageengine netflow analyzer professional 7, `http://www.manageengine.com/products/netflow/download.html`.

[159] P. Phall and M. Lavine. sflow version 5, chapter 6 security considerations. `http://sflow.org/sflow_version_5.txt`, July 2004.

[160] A. Jungmaier, E. Rescorla, and M. Tuexen. Transport Layer Security over Stream Control Transmission Protocol. RFC 3436, University of Essen, RTFM Inc., Siemens AG, December 2002.

[161] Nmap security scanner, `http://nmap.org/download.html`.

[162] Remi Badonnel, Radu State, and Olivier Festor. Management of mobile ad hoc networks: information model and probe-based architecture. *Int. J. Netw. Manag.*, 15(5):335–347, 2005.

[163] Allan Leinwand and Karen Fang. *Network management: a practical perspective*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[164] Wenli Chen, Nitin Jain, and S. Singh. Anmp: ad hoc network management protocol. *Selected Areas in Communications, IEEE Journal on*, 17(8):1506–1531, Aug 1999.

[165] D. Harrington, R. Presuhn, and B. Wijnen. An architecture for describing simple network management protocol (snmp) management frameworks, 2002.

[166] Chien-Chung Shen, C. Jaikaeo, C. Srisathapornphat, and Zhuochuan Huang. The guerrilla management architecture for ad hoc networks. In *MILCOM 2002. Proceedings*, volume 1, pages 467–472 vol.1, Oct. 2002.

[167] Kaustubh Suhas Phanse. *Policy-based quality of service management in wireless ad hoc networks*. PhD thesis, 2003. Chair-Dasilva, Luiz A.

[168] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[169] John Strassner. *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[170] A.M. Hadjiantonis, A. Malatras, and G. Pavlou. A context-aware, policy-based framework for the management of manets. In *Policies for Distributed Systems and Networks, 2006. Policy 2006. Seventh IEEE International Workshop on*, pages 10 pp.–34, June 2006.

[171] R. Chadha, Hong Cheng, Yuu-Heng Cheng, J. Chiang, A. Ghetie, G. Levin, and H. Tanna. Policy-based mobile ad hoc network management. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 35–44, June 2004.

[172] R. Chadha, Yuu-Heng Cheng, J. Chiang, G. Levin, Shih-Wei Li, and A. Poylisher. Policy-based mobile ad hoc network management for drama. In *Military Communications Conference, 2004. MILCOM 2004. IEEE*, volume 3, pages 1317–1323 Vol. 3, Oct.-3 Nov. 2004.

[173] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 2002.

[174] Cisco Systems. Performance Management: Best Practices White Paper. Technical Report http://www.cisco.com/application/pdf/paws/15115/perfmgmt.pdf, Cisco Systems, jan 2007.

[175] Brian Contos and Dave Kleiman. *Enemy at the Water Cooler: Real-Life Stories of Insider Threats and Enterprise Security Management Countermeasures*. Syngress Publishing, 2006.

[176] North Atlantic Treaty Organization. Concept of a NATO Security Management Infrastructure. Technical Report AC/322(SC/4-AHWG/3)WP(2007)0001, North Atlantic Treaty Organization, 2008.

[177] Guido v. d. Heidt and Reinhard Schoepf. Pki and entitlement —key information security management solutions for business and it compliance. *ISSE/SECURE 2007 Securing Electronic Business Processes*, pages 376–385, 2007.

[178] North Atlantic Treaty Organization. Security within the North Atlantic Treaty Organization (NATO). Technical Report C-M(2002)49, North Atlantic Treaty Organization, 2002.

[179] Thomas Bocek, Fabio Victora Hecht, Ela Hunt, David Hausheer, and Burkhard Stiller. Mobile P2P Fast Similarity Search. In *6th Annual IEEE Consumer Communications & Networking Conference (CCNC)*, Las Vegas, Nevada, USA, January 2009.

[180] Hercules Dalianis. Evaluating a spelling support in a search engine. In *NLDB*, pages 183–190, 2002.

[181] Reaz Ahmed and Raouf Boutaba. Distributed pattern matching: A key to flexible and efficient p2p search. *IEEE Journal on Selected Areas in Communications*, 25, Issue 1:73–83, January 2007.

[182] Bernard Wong, Aleksandrs Slivkins, and Emin Gn Sirer. Approximate Matching for Peer-to-Peer Overlays with Cubit. Technical Report http://hdl.handle.net/1813/10826, Cornell University, May 2008.

[183] Thomas Bocek, Ela Hunt, David Hausheer, and Burkhard Stiller. Fast Similarity Search in Peer-to-Peer Networks. In *11th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Salvador, Brazil, April 2008.

[184] Y. Xi, M. Chuah, and K. Chang. Performance Evaluation of a Power Management Scheme for DTNs. *Mobile Netw. Appl.*, 12(5):370–380, 2007.

[185] Hyewon Jun, Mostafa H. Ammar, Mark D. Corner, and Ellen W. Zegura. Hierarchical Power Management in Disruption Tolerant Networks with Traffic-Aware Optimization. In *Proceedings of the 2006 SIGCOMM Workshop on Challenged Networks, (CHANTS '06)*, pages 245–252, New York, NY, USA, September 2006. ACM Press.

[186] Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Yueng Hsieh. Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks. In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, (INFOCOM'02)*, volume 1, pages 200–209, 23-27 June 2002.

[187] Rong Zheng, Jennifer C. Hou, and Lui Sha. Asynchronous Wakeup for Ad Hoc Networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing, (MobiHoc '03)*, pages 35–45, New York, NY, USA, June 2003. ACM Press.

[188] Suresh Singh and C. S. Raghavendra. PAMAS: Power Aware Multi-Access Protocol with Signaling for Ad Hoc Networks. *ACM Computer Communication Review*, 28(3):5–26, July 1998.

[189] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of the IEEE INFOCOM*, pages 1567–1576, New York, NY, USA, June 2002.

[190] Hyewon Jun, M.H. Ammar, and E.W. Zegura. Power Management in Delay Tolerant Networks: A Framework and Knowledge-Based Mechanisms. In *IEEE SECON 2005 Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pages 418–429, 26-29 September 2005.

[191] N. Banerjee, M.D. Corner, and B.N Levine. An Energy-Efficient Architecture for DTN Throwboxes. In *Proceeding of IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 6-12 2007.

[192] C. M. Sadler and M. Martonosi. Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06)*, pages 265–278, New York, NY, USA, September 2006. ACM.

[193] Zhensheng Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *Communications Surveys & Tutorials, IEEE*, 8(1):24–37, 2006.

[194] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *Proceedings 25th IEEE International Conference on Computer Communications, (INFOCOM'06)*, pages 1–11, April 2006.

[195] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a Delay Tolerant Network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, (SIGCOMM '04)*, pages 145–158, New York, NY, USA, August 30 - September 03 2004. ACM Press.

[196] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic Routing in Intermittently Connected Networks. *SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, July 2003.

[197] Chipcon. Smartrf cc1000: Single chip very low power rf transceiver. http://www.chipcon.com.

[198] Albert F. Harris, III, Milica Stojanovic, and Michele Zorzi. When underwater acoustic nodes should sleep with one eye open: idle-time power management in underwater sensor networks. In *WUWNet '06: Proceedings of the 1st ACM international workshop on Underwater networks*, pages 105–108, Los Angeles, CA, USA, 2006.

[199] Matunda Nyanchama and Paul Sop. Enterprise security management: Managing complexity. *Information Systems Security*, 9(6):1–8, 2001.

[200] Heinz-Gerd Hegering, Sebastian Abeck, and Bernhard Neumair. *Integrated management of networked systems: concepts, architectures, and their operational application*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[201] Helge Janicke and Linda Finch. The role of dynamic security policy in military scenarios. In *Proc. 6th European Conference on Information Warfare and Security*, pages 121–130, July 2007.

[202] NSA. Enterprise security management: A context overview. 2.01.21.009, 2009.

[203] ISO/IEC. Information technology – Code of practice for information security management. Technical report, International Organization for Standardization/ International Electrotechnical Commission, December 2000.

[204] Tyrone Grandison, Marcel Bilger, L. O'Connor, M. Graf, Morton Swimmer, Matthias Schunter, Andreas Wespi, and Nev Zunic. Elevating the discussion on security management: The data centric paradigm. In Claudio Bartolini, Akhil Sahai, and Jacques Philippe Sauvé, editors, *BDIM*, pages 84–93. IEEE, 2007.

[205] Gabi Dreo Rodosek. *A Framework for IT Service Management*. PhD thesis, Ludwig?Maximilians?Universitt München, Habilitation Thesis, 2002.

[206] David Leadston. Enterprise Security Management Reducing the Pain of Managing Multiple IDS Systems. Technical report, SANS Institute, March 2004.

[207] Cristian Opincaru. *Service Oriented Security architecture applied to Spatial Data Infrastructures*. PhD thesis, Universität der Bundeswehr München, PhD Thesis, 2008.

[208] Symantec. Symantec enterprise security manager. http://www.symantec.com/business/enterprise-security-manager.

[209] IBM. Ibm tivoli. http://www-01.ibm.com/software/tivoli/.

[210] ArcSight. Arcsight enterprise security manager. http://www.arcsight.com/products/products-esm/.

[211] EMA. Ca esm. http://www.ca.com/files/IndustryAnalystReports/the_ema_all-stars_in_enterprise_systems.pdf.

[212] BMC Software. Bmc control-sa. http://www.bmc.com.

[213] e Security Inc. Open e-security platform (oesp) suite. http://www.esecurityinc.com.

[214] HP. Hp openview & security management. http://www.openview.hp.com/solutions/identity_mgt/sg/security_sg_jun03.pdf.

[215] Yoonsun Lim, Myung Kim, and Anmo Jeong. An enterprise security management system as an asp solution. In *ICHIT '06: Proceedings of the 2006 International Conference on Hybrid Information Technology*, pages 548–555, Washington, DC, USA, 2006. IEEE Computer Society.

[216] OASIS. Security assertion markup language (saml) v2.0. http://saml.xml.org/saml-specifications.

[217] Web services federation language (ws-federation). http://specs.xmlsoap.org/ws/2006/12/federation/.

[218] Wolfgang Hommel and Helmut Reiser. Federated identity management: Die notwendigkeit zentraler koordinationsdienste. In Paul Müller, Reinhard Gotzhein, and Jens B. Schmitt, editors, *KiVS Kurzbeiträge und Workshop*, volume 61 of *LNI*, pages 65–72. GI, 2005.

[219] OASIS. Key management interoperability protocol usage guide. http://xml.coverpages.org/ni2009-02-27-a.html#kmip-spec-v098.

[220] Security services markup language (s2ml). http://xml.coverpages.org/S2MLV08a.pdf.

[221] J. Linn. Generic Security Service Application Program Interface Version 2, Update 1. RFC 2743, RSA Laboratories, January 2000.

[222] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, SSH Communications Security Corp, Cisco Systems, January 2006.

[223] R. Enns. NETCONF Configuration Protocol. RFC 4741, Juniper Networks, December 2006.

[224] M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure SHell (SSH). RFC 4742, ThingMagic, ICEsoft Technologies, December 2006.

[225] D. Harrington and J. Schönwälder. Transport Subsystem for the Simple Network Management Protocol (SNMP). Internet Draft (work in progress) <draft-ietf-isms-tmsm-15.txt>, Huawei Technologies (USA), Jacobs University Bremen, November 2008.

[226] D. Harrington, J. Salowey, and W. Hardaker. Secure Shell Transport Model for SNMP. Internet Draft (work in progress) <draft-ietf-isms-secshell-13.txt>, Huawei Technologies, Cisco Systems, Sparta, November 2008.

[227] V. Marinov and J. Schönwälder. Performance Analysis of SNMP over SSH. In *Proc. 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006)*, number 4269 in LNCS, pages 25–36, Dublin, October 2006. Springer.

[228] J. Schönwälder and V. Marinov. On the Impact of Security Protocols on the Performance of SNMP. *(under review)*, 2008.

[229] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, SSH Communications Security Corp, Cisco Systems, January 2006.

[230] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252, SSH Communications Security Corp, Cisco Systems, January 2006.

[231] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254, SSH Communications Security Corp, Cisco Systems, January 2006.

[232] S. Lehtinen and C. Lonvick. The Secure Shell (SSH) Protocol Assigned Numbers. RFC 4250, SSH Communications Security Corp, Cisco Systems, January 2006.

[233] N. Provos, M. Friedl, and P. Honeyman. Preventing Privilege Escalation. In *Proc. 12th USENIX Security Symposium*, August 2003.

[234] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Independent, RTFM, April 2006.

[235] H. Shacham, D. Boneh, and E. Rescorla. Client-Side Caching for TLS. *ACM Transactions on Information and System Security*, 7(4):553–575, November 2004.

[236] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077, Cisco Systems, Nokia, Nokia Siemens Networks, January 2008.

[237] A. Kehne, J. Schönwälder, and H. Langendörfer. A Nonce-Based Protocol for Multiple Authentications. *Operating System Review*, 26(4):84–89, October 1992.

[238] A. Goldberg, R. Buff, and A. Schmitt. Secure Web Server Performance Dramatically Improved by Caching SSL Session Keys. In *Proc. Workshop on Internet Server Performance*, June 1998.

[239] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha. Securing Electronic Commerce: Reducing the SSL Overhead. *IEEE Network*, 14(4):8–16, July 2000.

[240] C. Coarfa, P. Druschel, and D. S. Wallach. Performance Analysis of TLS Web Servers. *ACM Transactions on Computer Systems*, 24(1), February 2006.

[241] T. Koponen, P. Eronen, and Mikko Saärelä. Resilient connections for SSH and TLS. In *Proc. of USENIX Annual Technical Conference 2006*, Boston, May 2006.

[242] L. Mamatas, S. Clayman, M. Charalambides, A. Galis, and G. Pavlou. Towards an information management overlay for the future internet. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010. To appear.

[243] Andy Cooke, Alasdair Gray, Lisha Ma, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Rob Byrom, Laurence Field, Steve Hicks, Jason Leake, Manish Soni, Antony Wilson, Roney Cordenonsi, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan, Stuart Kenny, and David O'callaghan. R-gma: An information integration system for grid monitoring. *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 462–481, 2003.

[244] R. Srinivasan. Xdr: External data representation standard, 1995.

[245] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, 30:2004, 2003.

[246] S. Clayman. Time indexing - an introduction. `http://www.timeindexing.com/documentation/papers.html`, 2003.

[247] A. Sahai, S. Graupner, V. Machiraju, and A. van Moorsel. Specifying and monitoring guarantees in commercial grids through sla. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 292–299, May 2003.

[248] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM, 2003.